A4

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 942 357 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
15.09.1999 Bulletin 1999/37

(51) Int. Cl.[6]: G06F 9/30

(21) Application number: 99104620.2

(22) Date of filing: 09.03.1999

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: 11.03.1998 JP 5968098
24.04.1998 JP 11558898

(71) Applicant:
Matsushita Electric Industrial Co., Ltd.
Kadoma-shi, Osaka 571-8501 (JP)

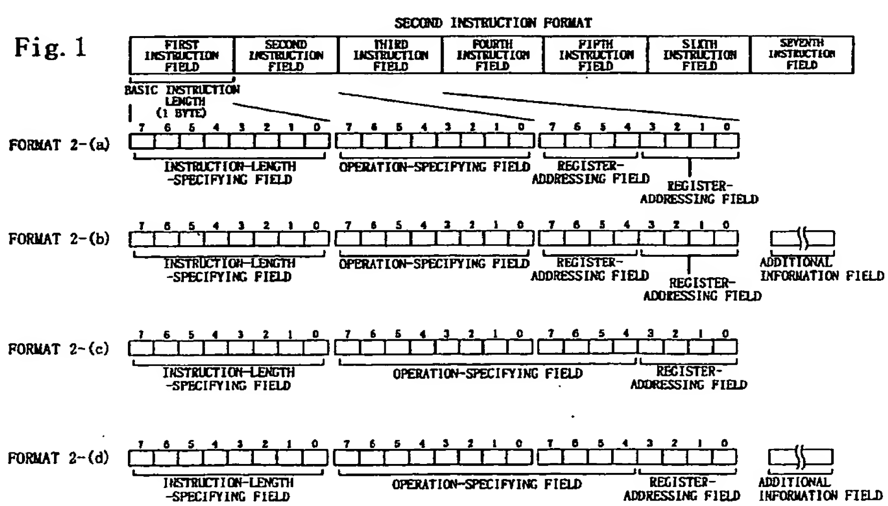
(72) Inventors:
- Kishida, Takeshi
Kashiwara-shi, Osaka 582-0028 (JP)
- Nakajima, Masaitsu
Osaka-shi, Osaka 536-0007 (JP)

(74) Representative:
Grünecker, Kinkeldey,
Stockmair & Schwanhäusser
Anwaltssozietät
Maximilianstrasse 58
80538 München (DE)

(54) Data processor compatible with a plurality of instruction formats

(57) A data processor according to the present invention executes instructions described in first and second instruction formats. The first instruction format defines a register-addressing field of a predetermined size, while the second instruction format defines a register-addressing field of a size larger than that of the register-addressing field defined by the first instruction format. The data processor includes: instruction-type identifier, responsive to an instruction, for identifying the received instruction as being described in the first or second instruction format by the instruction itself; a first register file including a plurality of registers; and a second register file also including a plurality of registers, the number of the registers included in the second register file being larger than that of the registers included in the first register file. If the instruction-type identifier has identified the received instruction as being described in the first instruction format, the data processor executes the instruction using data held in the first register file. On the other hand, if the instruction-type identifier has identified the received instruction as being described in the second instruction format, the data processor executes the instruction using data held in the second register file.

Fig. 1



EP 0 942 357 A2

Description

BACKGROUND OF THE INVENTION

[0001] The present invention relates to an improved data processor implemented as CPU or microprocessor, for example, and more particularly relates to a data processor adapted for use with an instruction set suitable for downsizing a program.

[0002] As semiconductor technologies and architecture of processors have been vigorously researched and developed over the past few years, the performance of programmable data processors of various types, termed "CPU's", have also been tremendously improved. Depending on their specific applications, CPU's may be roughly classified into the following two types: general-purpose microprocessors; and built-in microcontrollers for use in numerous kinds of control units and consumer electronics appliances. As for CPU's of the first type, that is, general-purpose microprocessors, improvement of performance is a top priority. Therefore, for the sake of improving the performance of general-purpose microprocessors, a wide variety of techniques have heretofore been employed. In contrast, with regards to built-in microcontrollers, it is true that improvement of performance is one of most significant objects to be achieved. What is more important is, however, striking an appropriate balance among performance improvement, cost effectiveness and reduction in power consumption. Among other things, the cost effectiveness plays a key role in meeting a high demand in consumer electronics applications.

[0003] There are two tips for realizing a CPU at a lower cost: reducing the size of a CPU itself (i.e., reducing the area of a CPU core); and shrinking the size of a program (or the size of a ROM). In recent years, as the performance of a CPU has been improved, the number of functions implementable by a single CPU has increased and the size of an application program has further increased correspondingly. Under the circumstances such as these, the size of a ROM for storing such a large-sized program dominates over the area of a CPU core. Accordingly, the cost effectiveness of a CPU is greatly dependent on how small the size of a program, applicable to the CPU, can be.

[0004] The prior art, developed to solve this task, will be described. In accordance with this technique, the architecture of an instruction set for a general-purpose microprocessor is extended to reduce the size of a program.

[0005] Figure 22 illustrates examples of MIPS architecture instruction format for a data processor in the pertinent prior art. Specifically, Figure **22(a)** illustrates a MIPS-II/III instruction format used for executing a register-to-register instruction where the length of a basic instruction word (hereinafter, simply referred to as a "basic instruction length") is fixed at 32 bits. In contrast, Figure **22(b)** illustrates MIPS16 instruction formats where the basic instruction length is fixed at 16 bits.

[0006] In accordance with the MIPS architecture, 32 registers are provided. Thus, an instruction set in the MIPS-II/III format includes a plurality of register-addressing fields each composed of 5 bits. Also, since three operands are specified according to the MIPS-II/III format, this instruction set includes three register-addressing fields **rs**, **rt** and **rd**. The operation and functions of the instruction are defined using a 6-bit OP field, a 5-bit shamt field and a 6-bit func field. Accordingly, this instruction set has a fixed length of 32 bits in total.

[0007] In contrast, two types of instruction formats are definable for a register-to-register instruction included in an instruction set according to the MIPS16 format. In one of the two types of instruction formats, two 3-bit register-addressing fields **rx** and **ry** are provided to specify two operands and the operation and function of the instruction are defined using a 5-bit OP field and a 5-bit func field. In the other instruction format, three 3-bit register-addressing fields **rx**, **ry** and **rz** are provided to specify three operands and the operation and function of the instruction are defined using a 5-bit OP field and a 2-bit F field.

[0008] In accordance with the MIPS16 format shown in Figure 22(b), only 3 bits are available for each register-addressing field. Accordingly, not all the 32 registers included in the original MIPS-II/III format, but some of these registers can be accessed.

[0009] Any instruction in the MIPS16 instruction format can be replaced with an associated instruction in the MIPS-II/III instruction format. Such replacement of an instruction in the MIPS16 instruction format with a counterpart in the MIPS-II/III instruction format is called an "extension" of an instruction.

[0010] Figure **23** is a block diagram illustrating a main part of a data processor for executing instructions in the MIPS16 and MIPS-II/III formats. Hereinafter, the operation of this data processor will be described.

[0011] An instruction fetch section **300** is a block for fetching an instruction. Specifically, the instruction fetch section **300** fetches an instruction set in the MIPS16 instruction format with a fixed length of 16 bits or in the MIPS-II/III instruction format with a fixed length of 32 bits, and then outputs the fetched instruction set to an instruction extender **310**. The type of the instruction set, i.e., whether the instruction set is in the MIPS16 or MIPS-II/III instruction format, is always specified by a mode setting signal.

[0012] The instruction extender **310** is also controlled by the mode setting signal. If the input instruction set is in the MIPS16 instruction format, then the instruction extender **310** extends the instruction set in the MIPS16 instruction format into that in the MIPS-II/III instruction format. Alternatively, if the input instruction set is in the MIPS-II/III instruction format, then the instruction extender **310** outputs the instruction set as it is without performing the extension. It is controlled by the mode setting signal whether or not the extension should be

performed. Accordingly, the instruction extender 310 cannot determine the necessity from the instructions themselves. Since the mode setting signal is a programmable signal, the mode of operations can be switched at a desired time.

[0013] An instruction decoder 320 is a block for decoding the instruction in the MIPS-II/III instruction format and thereby producing a control signal. The operation of the data processor is controlled by the control signal produced by the instruction decoder 320.

[0014] The data processor having such a configuration can execute both a program described in the MIPS16 instruction format with a basic instruction length of 16 bits and a program described in the MIPS-II/III instruction format with a basic instruction length of 32 bits. Accordingly, if the code size should be prioritized, then programming is preferably carried out using the MIPS16 instruction format with a fixed length of 16 bits. On the other hand, if the performance should be respected first to access as large a number of register files as possible, then programming may be conducted using the MIPS-II/III instruction format. Thus, a program can be developed flexibly with a good balance struck between performance and code size. Nevertheless, it depends sometimes on the specifications of a particular system and sometimes on the size of a program which type of instruction formats should be used. For example, a certain type of format is used only when the size of a program reaches that of a task.

[0015] In order for a microprocessor to perform such an application (like signal processing) as requiring a large number of registers, the number of available registers should preferably be increased by adding some registers to preexistent ones such that the application can be performed at an even higher speed. In such a case, an instruction format, allowing the user to specify a larger number of registers than a conventional instruction format, may be produced and used instead of the conventional instruction format. However, the size of a resultant program considerably increases by the use of such an alternate instruction format.

[0016] Thus, the prior art may be modified in the following manner. A new instruction format, allowing the user to specify a larger number of registers, may be provided in addition to the conventional instruction format. And the newly provided instruction format and the conventional instruction format may be selectively employed in response to the mode setting signal of the prior art.

[0017] Nevertheless, if the mode setting signal of the prior art is used, then the code size still increases disadvantageously. That is to say, a switching instruction should be given to generate the mode setting signal in switching the instruction formats. Accordingly, if a plurality of instructions, described in these formats, are included within a single instruction set, then the switching instructions should also be given numerous number of times, thus adversely increasing the code size.

SUMMARY OF THE INVENTION

[0018] An object of this invention is providing a data processor allowing for the use of additional registers to execute instructions in several types of instruction formats included within a single instruction set and to switch these formats without the mode setting signal while effectively reducing the code size.

[0019] To achieve this object, according to the present invention, a first instruction format, allowing the user to specify a number of registers, and a second instruction format, allowing the user to specify a larger number of registers than that of the registers specified in the first instruction format, are used. The types of instructions, described in these formats, are identifiable by the instructions themselves. Accordingly, an application requiring a large number of registers, like signal processing, can be performed at a higher speed without increasing the code size.

[0020] Specifically, a data processor according to the present invention executes an instruction described in a first instruction format and an instruction described in a second instruction format. The first instruction format defines a register-addressing field of a predetermined size, while the second instruction format defines a register-addressing field of a size larger than the size of the register-addressing field defined by the first instruction format. The data processor includes: means, responsive to an instruction, for identifying the received instruction as being described in the first or second instruction format by the instruction itself; a first register file including a plurality of registers; and a second register file also including a plurality of registers, the number of the registers included in the second register file being larger than the number of the registers included in the first register file. If the identifying means has identified the received instruction as being described in the first instruction format, the data processor executes the instruction using data held in the first register file. On the other hand, if the identifying means has identified the received instruction as being described in the second instruction format, the data processor executes the instruction using data held in the second register file.

[0021] In one embodiment of the present invention, the first instruction format defines a number of instruction fields and the second instruction format defines another number of instruction fields. And the identifying means identifies the received instruction as being described in the first or second instruction format by the contents of at least one of the instruction fields of the instruction that is defined by at least one predetermined ordinal number.

[0022] In another embodiment of the present invention, the number of the instruction fields defined by the second instruction format is larger than the number of the instruction fields defined by the first instruction format.

[0023] In still another embodiment, the predetermined

ordinal number of the instruction field used by the identifying means for format identification is first.

[0024] In still another embodiment, the second register file includes all of the registers included in the first register file.

[0025] In still another embodiment, the data processor further executes an instruction described in a third instruction format. The third instruction format specifies a plurality of operations and defines a register-addressing field of a size larger than that of the register-addressing field defined by the first instruction format. The register-addressing field defined by the third instruction format is used to specify one of the registers included in the second register file. Responsive to an instruction, the identifying means identifies the received instruction as being described in the third instruction format by the instruction itself.

[0026] Another data processor according to the present invention also executes an instruction described in a first instruction format and an instruction described in a second instruction format. The data processor includes: a register file including a predetermined number of registers, an address described in the first instruction format for specifying one of the registers being different from an address described in the second instruction format for specifying the same register; an address converter for receiving the instruction described in the first instruction format and converting an address described in the first instruction format, specified by the instruction to access one of the registers, into an address described in the second instruction format; and means, responsive to an instruction, for identifying the received instruction as being described in the first or second instruction format by the instruction itself. The output of the address converter is controlled by the output of the identifying means.

[0027] According to the present invention, an instruction format is provided for use in defining an arrangement of an instruction to be executed by a data processor. The instruction format is implemented as first and second instruction formats. The first instruction format defines a number of instruction fields and the second instruction format defines another number of instruction fields, the number of the instruction fields defined by the second instruction format being larger than the number of the instruction fields defined by the first instruction format. At least one of the instruction fields that are defined by the first and second instruction formats is used to identify the type of the instruction to be executed as being described in the first or second instruction format. The first instruction format defines a register-addressing field of a predetermined size, while the second instruction format defines a register-addressing field of a size larger than the size of the register-addressing field defined by the first instruction format.

[0028] In one embodiment of the present invention, the instruction format is implemented as a third instruction format. The third instruction format defines still another number of instruction fields, the number of the instruction fields defined by the third instruction format being larger than the number of the instruction fields defined by the first instruction format. The third instruction format defines a register-addressing field of a size larger than the size of the register-addressing field defined by the first instruction format. At least one of the instruction fields that are defined by the third instruction format is used to identify the type of the instruction to be executed as being described in the third instruction format. And the third instruction format describes a plurality of operations to be executed.

[0029] According to the present invention, the instruction itself is input to the identifying means, which identifies the instruction format thereof. In this case, the identifying means identifies the instruction format of the received instruction by the instruction itself, e.g., the contents of the first instruction field of the instruction. Accordingly, unlike the prior art, there is no need to use any special instruction to generate a mode setting signal or the like and the code size does not increase in vain. Accordingly, it is possible to effectively reduce the size of a program while allowing the user to execute a plurality of instructions described in several types of instruction formats included within a single instruction set.

[0030] In addition, in executing an instruction described in the first instruction format, a register to be accessed is specified from only a smaller number of registers included in the first register file. On the other hand, in executing an instruction described in the second instruction format, a register to be accessed can be specified from a larger number of registers included in the second register file. In this manner, arithmetic operations using these many registers can be described within a single instruction. Accordingly, the memory does not have to be accessed so frequently and data can be processed faster.

[0031] As can be understood, the present invention makes it possible to increase the number of usable registers and the speed of data processing while effectively reducing the size of a program.

[0032] Moreover, in accordance with the present invention, a plurality of operations can be specified within a single instruction described in the third instruction format. Accordingly, these operations, defined within a single instruction, can be performed in parallel, thus increasing the speed of data processing even more.

[0033] Furthermore, even if a bit assignment on the instruction code used to specify a register in the first instruction format is different from that used to specify the same register in the second instruction format, these bit assignments can be equalized through the address conversion by the address converter. Accordingly, complete compatibility can be maintained between a plurality of instruction formats, i.e., an

instruction set described in one of the instruction formats can be executed without rewriting the instruction set into another instruction format.

BRIEF DESCRIPTION OF THE DRAWINGS

[0034]

Figure **1** is a diagram illustrating a second instruction format for a data processor according to an exemplary embodiment of the present invention.
Figure **2** illustrates part of a list of instructions in the second instruction format to be executed by the data processor.
Figure **3** is a diagram illustrating a third instruction format for the data processor according to the embodiment of the present invention.
Figure **4** illustrates part of a list of instructions in the third instruction format to be executed by the data processor.
Figure **5** is a block diagram illustrating an overall arrangement of the data processor.
Figure **6** is a block diagram illustrating an arrangement registers in a register file of the data processor.
Figure **7** is a table of correspondence illustrating respective relationships among names, numbers and types of registers in the register file and associated bit assignments where the data processor executes instructions in the first instruction format.
Figure **8** is a table of correspondence illustrating respective relationships among names, numbers and types of registers in the register file and associated bit assignments where the data processor executes instructions in the second instruction format.
Figure **9** illustrates an example of a program written in C.
Figure **10** illustrates a first example where the program shown in Figure **9** is compiled in accordance with the first instruction format.
Figure **11** illustrates a second example where the program shown in Figure **9** is compiled in accordance with the first instruction format.
Figure **12** illustrates a third example where the program shown in Figure **9** is compiled in accordance with the first instruction format.
Figure **13** illustrates an example where the program shown in Figure **9** is compiled in accordance with the second instruction format.
Figure **14** illustrates an FIR filter processing program described without using instructions in the third instruction format according to the embodiment of the present invention.
Figure **15** illustrates a program obtained by rewriting the FIR filter processing program shown in Figure **14** in accordance with the third instruction format according to the embodiment of the present invention.
Figure **16** is a diagram illustrating a first instruction format (1) for the data processor according to the embodiment of the present invention.
Figure **17** illustrates part of a list of instructions in the first instruction format (1) to be executed by the data processor.
Figure **18** is a diagram illustrating a first instruction format (2) for the data processor.
Figure **19** illustrates part of a list of instructions in the first instruction format (2) to be executed by the data processor.
Figure **20** is a block diagram illustrating an arrangement of register in a first register file of the data processor.
Figure **21** is a table of correspondence illustrating respective relationships among names, numbers and types of registers in the register file and associated bit assignments where the data processor executes instructions in the first instruction format.
Figures **22(a)** and **22(b)** are diagrams illustrating exemplary instruction formats for a prior art data processor.
Figure **23** is a block diagram illustrating an arrangement of a main part of the prior art data processor around an instruction decoder.
Figure **24** is a diagram illustrating specific examples of first, second and third instruction formats used in the embodiment of the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0035] Hereinafter, preferred embodiments of the present invention will be described with reference to the accompanying drawings. Before the arrangement of the data processor of the present invention is described, three types of instruction formats used in the following embodiments will be described.

[0036] Figures **16** through **19** illustrate an outline of a first instruction format in accordance with the architecture of this embodiment.

[0037] In the first instruction format, a variable-length instruction with a minimum instruction length of 1 byte is described. A 2-bit field is used as a register-addressing field. Accordingly, four registers can be specified with one register-addressing field. In this architecture, four address registers and four data registers are defined. By separately using the address registers or the data registers responsive to a specific instruction, eight registers can be used in total in executing an instruction.

[0038] Figure 16 illustrates a bit assignment for the first instruction format (1) in which a first instruction field composed of 1 byte, equal to the minimum instruction length, consists of an operation-specifying field and an arbitrary number of register-addressing fields. Specific examples of this format will be described below.

[0039] In an exemplary first instruction format (1)-(a), the first instruction field includes two 2-bit register-

addressing fields and is composed of 1 byte, which is the minimum instruction length. And two operands can be specified in accordance with this format.

[0040] In another exemplary first instruction format (1)-(b), the first instruction field includes two 2-bit register-addressing fields, and an additional information field is further provided. Thus, the instruction length in accordance with this format is 2 bytes or more in total.

[0041] In still another exemplary first instruction format (1)-(c), the first instruction field includes one 2-bit register-addressing field and is composed of 1 byte, which is the minimum instruction length. And one operand can be specified in accordance with this format.

[0042] In yet another exemplary first instruction format (1)-(d), the first instruction field includes one 2-bit register-addressing field, and an additional information field is further provided. Thus, the instruction length in accordance with this format is 2 bytes or more in total.

[0043] In yet another exemplary first instruction format (1)-(e), the first instruction field includes no register-addressing fields and is composed of 1 byte, which is the minimum instruction length. Accordingly, in accordance with this format, no operands can be specified using addresses.

[0044] In yet another exemplary first instruction format (1)-(f), the first instruction field includes no register-addressing fields but an additional information field is further provided. Thus, the instruction length in accordance with this format is 2 bytes or more in total.

[0045] Figure 17 illustrates part of a list of specific instructions for respective types of bit assignment shown in Figure 16. In Figure 17, instruction mnemonics are shown on the left and the operations performed to execute these instructions are shown on the right.

[0046] Figure 18 illustrates a bit assignment for a first instruction format (2) in which a first instruction field composed of 1 byte, i.e., the minimum instruction length, consists of an instruction-length-specifying field and a second instruction field consists of an operation-specifying field and an arbitrary number of register-addressing fields. Specific examples of this format will be described in detail below.

[0047] In an exemplary first instruction format (2)-(a), the second instruction field includes two 2-bit register-addressing fields and the first and second instruction fields are composed of 2 bytes. And two operands can be specified in accordance with this format.

[0048] In another exemplary first instruction format (2)-(b), the second instruction field includes two 2-bit register-addressing fields, and an additional information field is further provided. Thus, the instruction length in accordance with this format is 3 bytes or more in total.

[0049] In still another exemplary first instruction format (2)-(c), the second instruction field includes one 2-bit register-addressing field and the first and second instruction fields are composed of 2 bytes. And one operand can be specified in accordance with this format.

[0050] In yet another exemplary first instruction format (2)-(d), the second instruction field includes one 2-bit register-addressing field, and an additional information field is further provided. Thus, the instruction length in accordance with this format is 3 bytes or more in total.

[0051] In yet another exemplary first instruction format (2)-(e), the second instruction field includes no register-addressing fields and the first and second instruction fields are composed of 2 bytes. Accordingly, in accordance with this format, no operands can be specified using addresses.

[0052] In yet another exemplary first instruction format (2)-(f), the second instruction field includes no register-addressing fields but an additional information field is further provided. Thus, the instruction length in accordance with this format is 3 bytes or more in total.

[0053] Figure 19 illustrates part of a list of specific instructions for respective types of bit assignment shown in Figure 18. In Figure 19, instruction mnemonics are shown on the left and the operations performed to execute these instructions are shown on the right.

[0054] Accordingly, in accordance with the first instruction format shown in Figures 16 through 19, the instruction length of the first instruction field is used as a basic instruction length to specify a variable-length instruction. And an instruction can be described in this format to have a length N times as large as the basic instruction length and equal to or less than the maximum instruction length, which is M times as large as the basic instruction length (where N and M are both positive integers and $1 \leq N \leq M$). Since the minimum instruction length is 1 byte, this instruction format is suitable for downsizing a program.

[0055] Figure 20 illustrates a first register file **220** included in the data processor of the present invention. The first register file **220** includes: four address registers **A0** through **A3**; four data registers **D0** through **D3**; a stack pointer (SP) **223**; a processor status word (PSW) **224** for holding internal status information and control information; and a program counter (PC) **225**. The first register file **220** further includes two read ports and one write port, thereby enabling reading from two registers (or reading from one register twice) and writing into one register simultaneously.

[0056] Figure 21 is a table illustrating accessing the address and data registers **A0** through **A3** and **D0** through **D3** included in the first register file **220** in greater detail. Specifically, this is a table of correspondence among name of a register specified by an instruction, bit assignment on an instruction code specified in a register-addressing field, and number and name of a physical register to be accessed.

[0057] In the first instruction format, the set of instruction addressing fields specified by respective instructions to access the four address registers **A0** through **A3** is the same as the set of instruction addressing fields specified by respective instructions to access the four data registers **D0** through **D3** as shown in Figure

21. That is to say, the same 2-bit instruction addressing field is used to address a desired register, and it is determined by the operation of the instruction itself whether an address register or a data register should be accessed.

[0058] Next, respective bit assignments for second and third instruction formats, which are added as respective extensions to the first instruction format shown in Figures 16 and 18, i.e., the basic instruction format of this architecture, will be described with reference to Figures 1 and 3.

[0059] In each of the bit assignments shown in Figure 1 for the second instruction format, a first instruction field, composed of 1 byte, which is the minimum instruction length, consists of an instruction-length-specifying field. And second and third instruction fields consist of an operation-specifying field and an arbitrary number of register-addressing fields. In accordance with the second instruction format, each register-addressing field is composed of 4 bits. Specific examples of this format will be described in detail below.

[0060] In an exemplary second instruction format (a), the third instruction field includes two 4-bit register-addressing fields and the first through third instruction fields are composed of 3 bytes in total. And two operands can be specified in accordance with this format.

[0061] In another exemplary second instruction format (b), the third instruction field includes two 4-bit register-addressing fields, and an additional information field is further provided. Thus, the instruction length in accordance with this format is 4 bytes or more in total.

[0062] In still another exemplary second instruction format (c), the third instruction field includes one 4-bit register-addressing field and the first through third instruction fields are composed of 3 bytes in total. And one operand can be specified in accordance with this format.

[0063] In yet another exemplary second instruction format (d), the third instruction field includes one 4-bit register-addressing field, and an additional information field is further provided. Thus, the instruction length in accordance with this format is 4 bytes or more in total.

[0064] Thus, in accordance with the second instruction format, the instruction length of first instruction field is also used as a basic instruction length. And an instruction can be described in this format to have a variable length N times as large as the basic instruction length and equal to or less than the maximum instruction length, which is M times as large as the basic instruction length (where N and M are both positive integers and $1 \leqq N \leqq M$).

[0065] Figure 2 illustrates part of a list of specific instructions for respective types of bit assignment shown in Figure 1. In Figure 2, instruction mnemonics are shown on the left and the operations performed to execute these instructions are shown on the right. The mnemonic Rm, Rn or Ri indicates the address of a specified register. In this case, any of sixteen general-purpose registers, namely, four address registers A0 through A3, four data registers D0 through D3 and eight extended registers E0 through E7, may be specified. Addressing of registers, as well as the configuration thereof, will be described in greater detail later.

[0066] In each of the bit assignments shown in Figure 3 for the third instruction format, a first instruction field, composed of 1 byte, which is the minimum instruction length, consists of an instruction-length-specifying field. A second instruction field consists of first and second operation-specifying fields, each composed of 4 bits. Each of third and fourth instruction fields consists of: a pair of 4-bit register-addressing fields: a pair of 4-bit operation-specifying fields; or a combination of one 4-bit register-addressing field and one 4-bit operation-specifying field. In accordance with the third instruction format, each register-addressing field is composed of four bits. In the following description, respective operations specified by the first and second operation-specifying fields, which are located in the second instruction field of the instruction described in this third instruction format, will be called "units".

[0067] Each unit is one of the instructions described in this third instruction format and corresponds to one of various instructions described in the second instruction format and used particularly frequently. In accordance with this third instruction format, the length of each operation-specifying field is shortened from eight bits in the second instruction format into four bits. Accordingly, two operations to be executed in parallel to each other can be described within a pair of operation-specifying fields. Thus, in accordance with this third instruction format, although the number of operations that can be described is limited, the code size thereof is smaller as compared with describing operations in the second instruction format.

[0068] In an exemplary third instruction format (a) shown in Figure 3, the total instruction length is four bytes. The second instruction field consists of first and second operation-specifying fields, each composed of four bits. Each of the third and fourth instruction fields includes a pair of 4-bit register-addressing fields. Thus, four operands can be specified in accordance with this instruction format.

[0069] In another exemplary third instruction format (b) shown in Figure 3, the total instruction length is also four bytes. The second instruction field also consists of first and second operation-specifying fields, each composed of four bits. The third instruction field consists of one 4-bit register-addressing field and a first operation-specifying field composed of four bits. And the fourth instruction field consists of a pair of 4-bit register-addressing fields. Thus, three operands can be specified in accordance with this instruction format.

[0070] In still another exemplary third instruction format (c) shown in Figure 3, the total instruction length is also four bytes. The second instruction field also consists of first and second operation-specifying fields,

each composed of four bits. The third instruction field consists of a pair of 4-bit register-addressing fields. And the fourth instruction field consists of one 4-bit register-addressing field and a second operation-specifying field composed of four bits. Thus, three operands can be specified in accordance with this instruction format.

[0071] In yet another exemplary third instruction format (d) shown in Figure 3, the total instruction length is also four bytes. The second instruction field also consists of first and second operation-specifying fields, each composed of four bits. The third instruction field consists of one 4-bit register-addressing field and a first operation-specifying field composed of four bits. And the fourth instruction field consists of one 4-bit register-addressing field and a second operation-specifying field composed of four bits. Thus, two operands can be specified in accordance with this instruction format.

[0072] In yet another exemplary third instruction format (e) shown in Figure 3, the total instruction length is also four bytes. The second instruction field also consists of first and second operation-specifying fields each composed of four bits. The third instruction field consists of a pair of 4-bit register-addressing fields. And the fourth instruction field consists of first and second operation-specifying fields each composed of four bits. Thus, two operands can be specified in accordance with this instruction format.

[0073] In this third instruction format, each of first and second units, specified by the first and second operation-specifying fields in the second instruction field, respectively, is an instruction described in the second instruction format (a) shown in Figure 2 or an instruction described in the second instruction format (d). In the latter case, the bit width of an immediate value is reduced to four bits.

[0074] Accordingly, in accordance with the third instruction format, the first instruction field specifies a variable-length instruction of 4 bytes, consisting of first through fourth instruction fields.

[0075] Figure 4 illustrates part of a list of specific instructions for respective types of bit assignment shown in Figure 3. In Figure 4, instruction mnemonics are shown on the left and the operations performed to execute these instructions are shown on the right. The mnemonic **Rm1**, **Rn1**, **Rm2** or **Rn2** indicates the address of a specified register. In this case, any of sixteen general-purpose registers, namely, four address registers **A0** through **A3**, four data registers **D0** through **D3** and eight extended registers **E0** through **E7**, may be specified. Also, **imm4** indicates a 4-bit immediate value. Furthermore, **Rm1** and **Rn1** are used to execute the first unit specified by the first operation-specifying field in the second instruction field, while **Rm2** and **Rn2** are used to execute the second unit specified by the second operation-specifying field in the second instruction field. For details, see the following description of operation.

[0076] Figure 5 is a block diagram illustrating an overall arrangement of a data processor according to an exemplary embodiment of the present invention.

[0077] The data processor has a five-stage pipelining structure consisting of: an instruction fetch (IF) stage; a decode and register read (DEC) stage; an execution (EX) stage; a memory access (MEM) stage; and a register write back (WB) stage.

[0078] As shown in Figure 5, the IF stage includes: an instruction memory **101**; an instruction fetch section **102**; and an instruction register **103**. The instruction memory **101** stores a program. The instruction fetch section **102** reads out a variable-length instruction, described in the first, second or third format, from the instruction memory **101**. And the instruction register **103** stores the variable-length instruction that has been read out via the instruction fetch section **102**.

[0079] The DEC stage includes an instruction decoder **110** for receiving and decoding the variable-length instruction that has been stored in the instruction register **103**.

[0080] Specifically, the instruction decoder **110** includes: an instruction-type identifier **111**; first, second and third register address extractors **112, 114** and **115**; a register address converter **113**; an operation decoder **116**; and first and second register address selectors **117** and **118**.

[0081] The instruction-type identifier **111** decodes the variable-length instruction stored in the instruction register **103** to identify the type of the instruction format, i.e., which of the first, second and third instruction formats. If the type of the variable-length instruction stored in the instruction register **103** is identified as the first instruction format, then the first register address extractor **112** extracts a 2-bit register-addressing field. The register address converter **113** converts the value of the register addressing field, extracted by the first register address extractor **112**, into a 4-bit register number in the register file. If the type of the variable-length instruction stored in the instruction register **103** is identified as the second instruction format, then the second register address extractor **114** extracts a 4-bit register-addressing field. And if the type of the variable-length instruction stored in the instruction register **103** is identified as the third instruction format, then the third register address extractor **115** extracts a 4-bit register-addressing field. Thereafter, the third register address extractor **115** supplies respective register addresses for the first and second units to the first and second register address selectors **117** and **118**, respectively.

[0082] The operation decoder **116** receives and decodes the variable-length instruction stored in the instruction register **103** and produces a control signal to be supplied to the EX stage. Based on the results of identification performed by the instruction-type identifier **111**, the first register address selector **117** selectively supplies the output of the register address converter **113**, the second register address extractor **114** or the third register address extractor **115** to a second register file **120**. The second register address selector **118**

selectively supplies either the output of the second register address extractor **114** or the output of the third register address extractor **115** to the second register file **120.** In this embodiment, the register address converter **113** always converts the extracted register address irrespective of the type of the instruction format. However, if the instruction-type identifier **111** can identify the type of instruction quickly, the address conversion may be naturally performed only on an instruction that has been identified as being described in the first instruction format.

[0083] The second register file **120** includes the first register file **220** shown in Figure **20** and is additionally provided with eight extended registers **E0** through **E7.** Accordingly, the second register file **120** includes sixteen general-purpose registers in total: **A0** through **A3;** **D0** through D3; and **E0** through **E7.** If the instruction in question is in the first or second instruction format, the addresses of these registers are input from the first register address selector **117** of the instruction decoder **110.** Alternatively, if the instruction in question is in the third instruction format, the addresses of these registers are input from the second register address selector **118.** The second register file **120** is different from the first register file **220** shown in Figure **20** in that the eight extended registers **E0** through **E7** are additionally provided for the second register file **120.** The other general-purpose registers, i.e., the four address registers **A0** through **A3** and the four data registers **D0** through **D3,** are used in common by these register files **120** and **220.**

[0084] The EX stage includes: E0, E1, E2 and E3 registers **131, 132, 134** and **135** for storing the respective outputs from the register file **120**; and operation units **133** and **136.** The operation unit **133** performs an arithmetic or logical operation using the contents of the E0 and E1 registers **131** and **132,** while the operation unit **136** performs an arithmetic or logical operation using the contents of the E2 and E3 registers **134** and **135.**

[0085] The MEM stage includes: M0 and M1 registers **141** and **142**; and a data memory **143.** The M0 and M1 registers **141** and **142** are pipeline registers for storing the outputs of the operation units **133** and **136,** respectively. The data memory **143** stores the data that has been stored in the M0 and M1 registers **141** and **142.**

[0086] And the WB stage includes W0 and W1 registers **151** and **152,** which are pipeline registers for storing the output of the data memory **143.**

[0087] Figure **6** illustrates the second register file **120** extracted from Figure **5.** And Figures **7** and **8** illustrate in greater detail accessing general-purpose registers included in this register file **120.**

[0088] Figure **7** is a table of correspondence among name of a register specified during the execution of an instruction defined in the first instruction format, bit assignment on an instruction code specified in a register-addressing field, and number and name of a physical register to be accessed. In accordance with the first instruction format, each register-addressing field is composed of only 2 bits. However, in this case, there are sixteen general-purpose registers, each of which should be accessed using a 4-bit address. Accordingly, address conversion should be performed. For example, in accessing an address register **A0** and a data register **D1,** "1000" and "1101" should be produced as respective physical register numbers and then output to a file **121** of general-purpose registers.

[0089] Figure **8** is a table of correspondence among name of a register specified during the execution of an instruction defined in the second instruction format, bit assignment on an instruction code specified in a register-addressing field, and number and name of a physical register to be accessed. In accordance with the second instruction format, each register-addressing field is composed of 4 bits, which is used as a physical register number as it is.

[0090] The operation of the data processor of the present invention having such a configuration will be described with reference to Figures **5** through **8.**

[0091] An instruction is given to the instruction-type identifier **111** shown in Figure **5.** In response thereto, the instruction-type identifier **111** identifies the type of the given instruction, i.e., which of the first, second and third instruction formats, by decoding a particular bit in the first instruction field. Then, the instruction-type identifier **111** produces a control signal associated with each of these formats. Hereinafter, it will be described in greater detail how the type of the instruction format is identified.

[0092] Figure **24** illustrates specific examples of instruction formats. As shown in Figure **24,** the first instruction format (1) may be implemented as **S0, S1, S2, S4** or **S6.** The instruction **S0** includes only a 1-byte operation code **OP** and has a minimum length of one byte. In contrast, the other instructions **S1, S2, S4** and **S6** additionally includes 8-, 16-, 32- or 48-bit immediate value **imm,** displacement **d** or absolute value **abs,** and are composed of 2, 3, 5 and 7 bytes, respectively. The first instruction format (2) may be implemented as **D0, D1, D2, D4** or **D5.** The instruction **D0** includes only a 2-byte operation code **OP** and has a minimum length of two bytes. In contrast, the other instructions **D1, D2, D4** and **D5** additionally includes 8-, 16-, 32- or 40-bit immediate value **imm,** displacement **d** or absolute value **abs,** and are composed of 3, 4, 6 and 7 bytes, respectively. The second instruction format may be implemented as **T0, T1, T3** or **T4.** The instruction **T0** includes only a 3-byte operation code **OP** and has a minimum length of three bytes. In contrast, the other instructions **T1, T3** and **T4** additionally includes 8-, 24- or 32-bit immediate value **imm,** displacement **d** or absolute value **abs,** and are composed of 4, 6 and 7 bytes, respectively. The third instruction format is herein implemented as an instruction **Q0** and is composed of only a 4-byte operation code **OP.** In the instruction formats shown in Figure **24,** judging only from the number of operation codes **OP,** the number of operation fields (i.e., three) of the

instructions **T0** through **T4** in the second instruction format is larger than that of the instructions **S0** through **S6** in the first instruction format (1) (i.e., one) or that of the instructions **D0** through **D5** in the first instruction format (2) (i.e., two). Also, the number of instruction fields (i.e., four) of the instruction **Q0** in the third instruction format is larger than that (i.e., three) of the instructions **T0** through **T4** in the second instruction format. The first instruction formats (1) and (2) are assigned to instructions that can be described in a relatively small number of fields and are specified frequently. On the other hand, the second instruction format is assigned to instructions that are described in a relatively large number of fields and are specified less frequently.

[0093] In the variable-length instructions shown in Figure **24**, the bit codes are assigned to respective first bytes as shown on the right column in Figure **24** depending on the respective types **S0** through **S6, D0** through **D5, T0** through **T4** and **Q0.** Accordingly, by recognizing the bit codes in the first byte, the type of the received instruction can be uniquely identified as being first, second or third instruction format. In this embodiment, the type of the instruction format is identified only by the bit codes in the first byte. Alternatively, the type of the instruction format may be identified according to the present invention by: the fourth bit of the second byte (i.e., the most significant bit of the lower nibble) in addition to the first byte; part of the first byte, not all of it; or part or all of a byte that is defined by a predetermined ordinal number.

[0094] First, it will be exemplified how an instruction **MOV A0, D0** in the first instruction format is executed as well as individual times of operations. This is a register-to-register transfer instruction that the contents of the address register **A0** should be read out and then stored in the data register **D0.**

(a) IF stage

[0095] The instruction fetch section **102** reads out the instruction from the instruction memory **101** and then stores the instruction in the instruction register **103.**

(b) DEC stage

[0096] The instruction-type identifier **111** decodes the instruction stored in the instruction register **103** to identify the type of this instruction as the first instruction format and output a control signal indicative of the type to the first register address selector **117.** The first register address extractor **112** extracts a source register address "00", which is the address of the address register **A0,** and a destination register address "00", which is the address of the data register **D0.** The register address converter **113** converts the address "00" of the address register **A0** and the address "00" of the data register **D0** into 4-bit addresses "1000" and "1100", respectively. Responsive to the control signal supplied from the instruction-type identifier **111,** the first register address selector **117** outputs the addresses, supplied from the register address converter **113,** to the register file **120.** The operation decoder **116** decodes the instruction as a register-to-register transfer instruction from the address register **A0** to the data register **D0,** thereby producing an associated control signal.

[0097] The contents of the address register **A0,** associated with the physical address "1000", are read out as a source operand from the register file **120** and then stored in the E0 register **131.**

(c) EX stage

[0098] The data stored in the E0 register **131** is passed through the operation unit **133** and then stored in the M0 register **141.**

(d) MEM stage

[0099] The data stored in the M0 register **141** is passed through the data memory **143** and then stored in the W0 register **151.**

(e) WB stage

[0100] The data stored in the W0 register **151** is written into the data register **D0** associated with the physical address "1100".

[0101] The above-described operations enable the execution of an instruction in the first instruction format.

[0102] Next, it will be exemplified how an instruction **MOV A0, E7** in the second instruction format is executed as well as individual times of operations. This is a register-to-register transfer instruction that the contents of the address register **A0** should be read out and then stored in the extended register **E7.**

(a) IF stage

[0103] The instruction fetch section **102** reads out the instruction from the instruction memory **101** and then stores the instruction in the instruction register **103.**

(b) DEC stage

[0104] The instruction-type identifier **111** decodes the instruction stored in the instruction register **103** to identify the type of this instruction as the second instruction format and output a control signal indicative of the type to the first register address selector **117.** The second register address extractor **114** extracts a source register address "1000", which is the address of the address register **A0,** and a destination register address "0111", which is the address of the extended register **E7.** Responsive to the control signal supplied from the instruction-type identifier **111,** the first register address selector **117** outputs the addresses, supplied from the

second register address extractor **114,** to the register file **120.** The operation decoder **116** decodes the instruction as a register-to-register transfer instruction from the address register **A0** to the extended register **E7,** thereby producing an associated control signal.

[0105] The contents of the address register **A0,** associated with the physical address "1000", are read out as a source operand from the register file **120** and then stored in the E0 register **131.**

(c) EX stage

[0106] The data stored in the E0 register **131** is passed through the operation unit **133** and then stored in the M0 register **141.**

(d) MEM stage

[0107] The data stored in the M0 register **141** is passed through the data memory **143** and then stored in the W0 register **151.**

(e) WB stage

[0108] The data stored in the W0 register **151** is written into the extended register **E7** associated with the physical address "0111".

[0109] The above-described operations enable the execution of an instruction in the second instruction format.

[0110] Next, it will be exemplified how an instruction **MOV_MOV E5, E6, E4, E7** in the third instruction format is executed, as well as individual times of operations. This is a parallel register-to-register transfer instruction that the contents of the extended register **E5** should be read out and stored in the extended register **E6,** and at the same time, the contents of the extended register **E4** should be read out and stored in the extended register **E7.**

(a) IF stage

[0111] The instruction fetch section **102** reads out the instruction from the instruction memory **101** and then stores the instruction in the instruction register **103.**

(b) DEC stage

[0112] The instruction-type identifier **111** decodes the instruction stored in the instruction register **103** to identify the type of this instruction as the third instruction format and output a control signal indicative of the type to the first register address selector **117.**

[0113] Following the first transfer instruction, the third register address extractor **115** extracts a source register address "0101", which is the address of the extended register **E5,** and a destination register address "0110", which is the address of the extended register **E6.** In addition, following the second transfer instruction, the third register address extractor **115** also extracts a source register address "0100", which is the address of the extended register **E4,** and a destination register address "0111", which is the address of the extended register **E7.**

[0114] Responsive to the control signal supplied from the instruction-type identifier **111,** the first register address selector **117** outputs the first and second sets of addresses, supplied from the third register address extractor **115,** to the register file **120.**

[0115] The operation decoder **116** decodes the instruction as a parallel register-to-register transfer instruction **VLIW** from the extended register **E5** to the extended register **E6** and from the extended register **E4** to the extended register **E7,** thereby producing an associated control signal.

[0116] The contents of the extended register **E5,** associated with the physical address "0101", are read out as a first source operand from the register file **120** and then stored in the E0 register **131.** At the same time, the contents of the extended register **E4,** associated with the physical address "0100", are read out as a second source operand from the register file **120** and then stored in the E2 register **134.**

(c) EX stage

[0117] The data stored in the E0 register **131** is passed through the operation unit **133** and then stored in the M0 register **141.** Simultaneously, the data stored in the E2 register **134** is passed through the operation unit **136** and then stored in the M1 register **142.**

(d) MEM stage

[0118] The data stored in the M0 register **141** is passed through the data memory **143** and then stored in the W0 register **151.** Concurrently, the data stored in the M1 register **142** is passed through the data memory **143** and then stored in the W1 register **152.**

(e) WB stage

[0119] The data stored in the W0 register **151** is written into the extended register **E6** associated with the physical address "0110". At the same time, the data stored in the W1 register **152** is written into the extended register **E7** associated with the physical address "0111".

[0120] The above-described operations enable the execution of an instruction in the third instruction format.

[0121] As can be understood from the foregoing description, the data processor of the present invention can execute any instruction without switching modes, no matter which of the first, second and third formats the instruction is described in. Also, the data processor of the present invention can execute an instruction set,

which has been optimized for reducing the code size, with its upward compatibility completely maintained and with the number of available registers considerably increased.

[0122] Next, specific examples showing how instructions in the first instruction format are made compatible with counterparts in the second instruction format will be described.

[0123] Figure 9 illustrates a brief example of a program written in C. This is a simple instruction that the contents of a variable **a** should be added to the contents of a variable **b** and the sum should be stored as a new variable **a**, and that the contents of a variable **c** should be added to the contents of a variable **d** and the sum should be stored as a new variable **c**.

[0124] The exemplary results obtained by compiling this program into instructions in the first instruction format are shown in Figures **10, 11** and **12**.

[0125] Figure **10** illustrates an instruction set where all the variables are assigned to the data registers. Specifically, the variables **a, b, c** and **d** are supposed to have been successfully assigned to the data registers **D0, D1, D2** and **D3**, respectively. Such a state is feasible by executing only two addition instructions. However, since there are only four data registers, it is difficult to assign all of these variables to the same number of registers.

[0126] Figure **11** illustrates an instruction set where the respective variables are assigned to stack regions on a data memory. Specifically, the variables **a, b, c** and **d** are supposed to have been successfully assigned to (SP+#4), (SP+#8), (SP+#12) and (SP+#16), respectively. In this case, not only the two addition instructions, but also eight data transfer instructions between memory and registers, i.e., Instructions 1 through 8, should be executed. As a result, the processing performance of the data processor considerably deteriorates.

[0127] Figure **12** illustrates an instruction set further including data transfer instructions, i.e., Instructions 1, 2, 11 and 12, to save and restore the contents of the data registers **D0** and **D1** used for addition. In such a case, only non-destructive registers can be used.

[0128] In contrast, Figure 13 illustrates an instruction set where all the variables are assigned to extended registers in accordance with the second instruction format. Specifically, the variables **a, b, c** and **d** are supposed to have been successfully assigned to the extended registers **E0, E1, E2** and **E3**, respectively. Such a state is feasible by executing two addition instructions. Since the number of extended registers is also limited, a sufficiently large number of registers are not always available from the extended registers. However, in this case, the number of registers available is much larger than the case shown in Figure **10**, the performance of the data processor is less likely to deteriorate due to frequent access to the memory. Accordingly, in order to execute an application program, requiring a large number of registers as in signal processing, in as short a time as possible, the extended registers **E0** through **E7** should be added and the second instruction format should be used. In such a case, the number of times the memory is accessed can be reduced and the processing performance can be improved.

[0129] In addition, even if an address specified for a certain register in the first instruction format is different from an associated address specified for the same register in the second instruction format, the first and second instruction formats can be used interchangeably in this embodiment. This is because the address specified in the first instruction format can be converted into the address specified in the second instruction format. Accordingly, even if the number of registers is increased by the addition of extended registers, the first and second instruction formats can be used with upward compatibility completely maintained. Consequently, the present invention is particularly advantageous in that this data processor can cope with the addition of registers flexibly.

[0130] Next, specific examples of how to execute instructions in the third instruction format will be described.

[0131] Figure **14** illustrates a main part of a program for an FIR filter that processes a medium such as a software-driven modem. In Figure **14**, the program is described without using instructions in the third instruction format, and the code sizes of respective instructions are shown on the right of associated operands.

[0132] Instruction 1 represents a half-word transfer instruction from a register to a memory (store) in the first instruction format. Instruction 2 represents a half-word swap instruction (swap) in the second instruction format. Instruction 3 represents a half-word transfer instruction from a register to a memory (store) in the second instruction format. Instruction 4 represents a dual half-word multiply-and-accumulate instruction in the second instruction format. Instruction 5 represents a subtraction instruction in the second instruction format. Instructions 6 and 7 each represent a transfer instruction from a memory to a register (load) in the second instruction format. And Instruction 8 represents a conditional branch instruction in the first instruction format.

[0133] Hereinafter, specific times of operations will be described.

[0134] Instruction 1 **MOVH D0, (#-6, A3)** is a register-to-memory transfer instruction (store) in the first instruction format that a value in the data register **D0** should be stored at a memory address obtained by subtracting "6" from an address stored in the address register **A3**.

(a) IF stage

[0135] The instruction fetch section **102** reads out the instruction from the instruction memory **101** and then stores the instruction in the instruction register **103**.

(b) DEC stage

[0136] The instruction-type identifier **111** decodes the instruction stored in the instruction register **103** to identify the type of this instruction as the first instruction format and output a control signal indicative of the type to the first register address selector **117.** The first register address extractor **112** extracts source register addresses "00" and "11", which are the respective addresses of the data register **D0** and the address register **A3.** The register address converter **113** converts the address "00" of the data register **D0** and the address "11" of the address register **A3** into 4-bit addresses "1100" and "1011", respectively.

[0137] Responsive to the control signal supplied from the instruction-type identifier **111**, the first register address selector **117** outputs the respective 4-bit addresses "1100" and "1011" of the data and address registers **D0** and **A3,** supplied from the register address converter **113,** to the register file **120.**

[0138] The operation decoder **116** decodes the instruction as a register-to-memory transfer instruction (store) that a value in the data register **D0** should be stored at a memory address obtained by subtracting "6" from an address stored in the address register **A3,** thereby producing an associated control signal.

[0139] The contents of the data register **D0,** associated with the physical address "1100", are read out as a source operand from the register file **120** and then stored in the E0 register **131.** And the contents of the address register **A3,** associated with the physical address "1011", are read out as another source operand from the register file **120** and then stored in the E1 register **132.**

(c) EX stage

[0140] The data stored in the E0 register **131** is passed through the operation unit **133** and then stored in the M0 register **141.** In response to the control signal supplied from the operation decoder **116,** the operation unit **133** subtracts "6" from the address stored in the E1 register **132** and outputs the result of subtraction to an address input section of the data memory **143.**

(d) MEM stage

[0141] The data stored in the M0 register **141** is stored in the data memory **143** at an address specified by the result of subtraction performed by the operation unit **133.**

(e) WB stage

[0142] No operations are performed.

[0143] Instruction 2 **SWHW E0, E2** is an instruction in the second instruction format that data, obtained by swapping the upper and lower half words of the data stored in the extended register **E0,** should be stored in the extended register **E2.**

(a) IF stage

[0144] The instruction fetch section **102** reads out the instruction from the instruction memory **101** and then stores the instruction in the instruction register **103.**

(b) DEC stage

[0145] The instruction-type identifier **111** decodes the instruction stored in the instruction register **103** to identify the type of this instruction as the second instruction format and output a control signal indicative of the type to the first register address selector **117.** The second register address extractor **114** extracts a source register address "0000", which is the address of the extended register **E0,** and a destination register address "0010", which is the address of the extended register **E2.**

[0146] Responsive to the control signal supplied from the instruction-type identifier **111,** the first register address selector **117** outputs the addresses, supplied from the second register address extractor **114,** to the register file **120.** The operation decoder **116** decodes the instruction to find that data, obtained by swapping the upper and lower half words of the data stored in the extended register **E0,** should be stored in the extended register **E2,** thereby producing an associated control signal.

[0147] The contents of the extended register **E0,** associated with the physical address "0000", are read out as a source operand from the register file **120** and then stored in the E0 register **131.**

(c) EX stage

[0148] Responsive to the control signal supplied from the operation decoder **116,** the operation unit **133** performs a swapping operation on the data stored in the E0 register **131** by swapping the upper and lower half words thereof. Then, the result of this operation is stored in the M0 register **141.**

(d) MEM stage

[0149] The data stored in the M0 register **141** is passed through the data memory **143** and then stored in the W0 register **151.**

(e) WB stage

[0150] The data stored in the W0 register **151** is read out and then stored in the register file **120** at the destination address "0010" specified by the second register address extractor **114.**

[0151] Instruction 3 **MOVH E2, (#-4, A3)** is a register-to-memory transfer instruction (store) in the second

instruction format that a value in the extended register **E2** should be stored at a memory address obtained by subtracting "4" from an address stored in the address register **A3**.

(a) IF stage

[0152] The instruction fetch section **102** reads out the instruction from the instruction memory **101** and then stores the instruction in the instruction register **103**.

(b) DEC stage

[0153] The instruction-type identifier **111** decodes the instruction stored in the instruction register **103** to identify the type of this instruction as the second instruction format and output a control signal indicative of the type to the first register address selector **117**. The first register address extractor **112** extracts a source register address "11", which is the address of the address register **A3**. The second register address extractor **114** extracts another source register address "0010", which is the address of the extended register **E2**. The register address converter **113** converts the address "11" of the address register **A3** into a 4-bit address "1011".

[0154] Responsive to the control signal supplied from the instruction-type identifier **111**, the first register address selector **117** outputs the 4-bit address "1011" of the address register **A3**, supplied from the register address converter **113**, and the 4-bit address "0010" of the extended register **E2** to the register file **120**.

[0155] The operation decoder **116** decodes the instruction as a register-to-memory transfer instruction (store) that a value in the extended register E2 should be stored at a memory address obtained by subtracting "4" from an address stored in the address register **A3**, thereby producing an associated control signal.

[0156] The contents of the extended register **E2**, associated with the physical address "0010", are read out as a source operand from the register file **120** and then stored in the E0 register **131**. And the contents of the address register **A3**, associated with the physical address "1011", are read out as another source operand from the register file **120** and then stored in the E1 register **132**.

(c) EX stage

[0157] The data stored in the E0 register **131** is passed through the operation unit **133** and then stored in the M0 register **141**. In response to the control signal supplied from the operation decoder **116**, the operation unit **133** subtracts "4" from the address stored in the E1 register **132** and outputs the result of subtraction to an address input section of the data memory **143**.

(d) MEM stage

[0158] The data stored in the M0 register **141** is stored in the data memory **143** at the address specified by the result of subtraction performed by the operation unit **133**.

(e) WB stage

[0159] No operations are performed.

[0160] Instruction 4 **DMACH E6, E1** is an instruction in the second instruction format that a product obtained by multiplying together the respective upper half words of the data stored in the extended registers **E6** and **E1** should be added to a product obtained by multiplying together the respective lower half words thereof and the sum should be accumulated in the extended register **E1**.

(a) IF stage

[0161] The instruction fetch section **102** reads out the instruction from the instruction memory **101** and then stores the instruction in the instruction register **103**.

(b) DEC stage

[0162] The instruction-type identifier **111** decodes the instruction stored in the instruction register **103** to identify the type of this instruction as the second instruction format and output a control signal indicative of the type to the first register address selector **117**. The second register address extractor **114** extracts a source register address "0110", which is the address of the extended register **E6**, and a destination register address "0001", which is the address of the extended register **EI**.

[0163] Responsive to the control signal supplied from the instruction-type identifier **111**, the first register address selector **117** outputs the addresses, supplied from the second register address extractor **114**, to the register file **120**. The operation decoder **116** decodes the instruction to find that a product obtained by multiplying together the respective upper half words of the data stored in the extended registers **E6** and **E1** should be added to a product obtained by multiplying together the respective lower half words thereof and the sum should be accumulated in the extended register **E1**, thereby producing an associated control signal.

[0164] The contents of the extended register **E6**, associated with the physical address "0110", are read out as a source operand from the register file **120** and then stored in the E0 register **131**. And the contents of the extended register **E1**, associated with the physical address "0001", are read out as another source operand from the register file **120**.

(c) EX stage

[0165] The operation unit **133** extracts and multiplies together the respective upper half words of the data stored in the E0 and E1 registers **131** and **132**. At the same time, the operation unit **133** extracts and multiplies together the respective lower half words of the data stored in the E0 and E1 registers **131** and **132**. These two products are added to the data stored in the E1 register **132**. And the result of addition is stored in the M0 register **141**.

(d) MEM stage

[0166] The data stored in the M0 register **141** is passed through the data memory **143** and then stored in the W0 register **151**.

(e) WB stage

[0167] The data stored in the W0 register **151** is read out and then stored in the register file **120** at the destination address "0001" specified by the second register address extractor **114**.

[0168] Instruction 5 **SUB 1, E3** is a subtraction instruction in the second instruction format that "1" should be subtracted from the value stored in the extended register E3 and then the result of subtraction should be stored in the extended register **E3**.

(a) IF stage

[0169] The instruction fetch section **102** reads out the instruction from the instruction memory **101** and then stores the instruction in the instruction register **103**.

(b) DEC stage

[0170] The instruction-type identifier **111** decodes the instruction stored in the instruction register **103** to identify the type of this instruction as the second instruction format and output a control signal indicative of the type to the first register address selector **117**. The second register address extractor **114** extracts a source register address "0011", which is the address of the extended register **E3**, and a destination register address "0011", which is also the address of the extended register **E3**.

[0171] Responsive to the control signal supplied from the instruction-type identifier **111**, the first register address selector **117** outputs the addresses, supplied from the second register address extractor **114**, to the register file **120**. The operation decoder **116** decodes the instruction to find that "1" should be subtracted from the value stored in the extended register **E3**, thereby producing an associated control signal.

[0172] The contents of the extended register **E3**, associated with the physical address "0011", are read out as a source operand from the register file **120** and then stored in the E0 register **131**.

(c) EX stage

[0173] The operation unit **133** reads out the data stored in the E0 register **131** and subtracts "1" therefrom. And the result of subtraction is stored in the M0 register **141**.

(d) MEM stage

[0174] The data stored in the M0 register **141** is passed through the data memory **143** and then stored in the W0 register **151**.

(e) WB stage

[0175] The data stored in the W0 register **151** is read out and then stored in the register file **120** at the destination address "0011" specified by the second register address extractor **114**.

[0176] Instruction 6 **MOV (#4, E4+), E6** is a post-increment memory-to-register transfer instruction in the second instruction format that data should be read out from the memory address of the extended register **E4** and then stored in the extended register **E6** and the value of the extended register **E4** should be increased by "4" after the storage.

(a) IF stage

[0177] The instruction fetch section **102** reads out the instruction from the instruction memory **101** and then stores the instruction in the instruction register **103**.

(b) DEC stage

[0178] The instruction-type identifier **111** decodes the instruction stored in the instruction register **103** to identify the type of this instruction as the second instruction format and output a control signal indicative of the type to the first register address selector **117**. The second register address extractor **114** extracts a source register address "0100", which is the address of the extended register **E4**, and a destination register address "0110", which is the address of the extended register **E6**.

[0179] Responsive to the control signal supplied from the instruction-type identifier **111**, the first register address selector **117** outputs the addresses, supplied from the second register address extractor **114**, to the register file **120**. The operation decoder **116** decodes the instruction as a post-increment memory-to-register transfer instruction that data should be read out from a memory address of the extended register **E4** and then stored in the extended register **E6** and the value of the extended register **E4** should be increased by "4" after the storage, thereby producing an associated control signal.

[0180] The contents of the extended register **E4**, associated with the physical address "0100", are read out as a source operand from the register file **120** and then stored in the E0 register **131**.

(c) EX stage

[0181] The operation unit **133** reads out the data stored in the E0 register **131** and outputs the data to an address input section of the data memory **143** in response to the control signal supplied from the operation decoder **116**. Also, the operation unit **133** adds "4" to the data read out, and stores the sum in the M0 register **141**.

(d) MEM stage

[0182] The data stored in the M0 register **141** is passed through the data memory **143** and then stored in the W0 register **151**. Also, responsive to the control signal supplied from the operation decoder **116**, data is read out from the data memory **143** at the specified memory address and then stored in the W0 register **151**.

(e) WB stage

[0183] The data stored in the W0 register **151** is read out and then stored in the register file **120** at the destination address "0110" specified by the second register address extractor **114**.

[0184] Instruction 7 **MOV (#4, E5+), E1** is a post-increment memory-to-register transfer instruction in the second instruction format that data should be read out from the memory address of the extended register **E5** and then stored in the extended register **E1** and the value of the extended register **E5** should be increased by "4" after the storage.

(a) IF stage

[0185] The instruction fetch section **102** reads out the instruction from the instruction memory **101** and then stores the instruction in the instruction register **103**.

(b) DEC stage

[0186] The instruction-type identifier **111** decodes the instruction stored in the instruction register **103** to identify the type of this instruction as the second instruction format and output a control signal indicative of the type to the first register address selector **117**. The second register address extractor **114** extracts a source register address "0101", which is the address of the extended register **E5**, and a destination register address "0001", which is the address of the extended register **E1**.

[0187] Responsive to the control signal supplied from the instruction-type identifier **111**, the first register address selector **117** outputs the addresses, supplied from the second register address extractor **114**, to the register file **120**. The operation decoder **116** decodes the instruction as a post-increment memory-to-register transfer instruction that data should be read out from the memory address of the extended register **E5** and then stored in the extended register **E1** and the value of the extended register **E5** should be increased by "4" after the storage, thereby producing an associated control signal.

[0188] The contents of the extended register **E5**, associated with the physical address "0101", are read out as a source operand from the register file **120** and then stored in the E0 register **131**.

(c) EX stage

[0189] The operation unit **133** reads out the data stored in the E0 register **131** and outputs the data to an address input section of the data memory **143** in response to the control signal supplied from the operation decoder **116**. Also, the operation unit **133** adds "4" to the data read out, and stores the sum in the M0 register **141**.

(d) MEM stage

[0190] The data stored in the M0 register **141** is passed through the data memory **143** and then stored in the W0 register **151**. Also, responsive to the control signal supplied from the operation decoder **116**, data is read out from the data memory **143** at the specified memory address and then stored in the W0 register **151**.

(e) WB stage

[0191] The data stored in the W0 register **151** is read out and then stored in the register file **120** at the destination address "0001" specified by the second register address extractor **114**.

[0192] Instruction 8 **LGE** is an instruction in the first instruction format that if the result of subtraction performed to execute Instruction 4 is equal to or larger than "0", then a loop instruction should be executed by changing the value of the program counter **124** into an address **LOOP** shown in Figure **14** above Instruction 1.

(a) IF stage

[0193] The instruction fetch section **102** reads out the instruction from the instruction memory **101** and then stores the instruction in the instruction register **103**.

(b) DEC stage

[0194] The instruction-type identifier **111** decodes the instruction stored in the instruction register **103** to identify

tify the type of this instruction as the first instruction format and output a control signal indicative of the type to the first register address selector **117**. The operation decoder **116** decodes the instruction as a loop instruction, thereby producing an associated control signal.

[0195] Responsive to the control signal supplied from the operation decoder **116**, the value of the program counter **124** in the register file **120** is stored in the E0 register **131**.

(c) EX stage

[0196] The operation unit **133** reads out the data stored in the E0 register **131** and stores the destination address of the loop in the M0 register **141** responsive to the control signal supplied from the operation decoder **116**.

(d) MEM stage

[0197] The data stored in the M0 register **141** is passed through the data memory **143** and then stored in the W0 register **151**.

(e) WB stage

[0198] The data stored in the W0 register **151** is stored at the program counter **124** in the register file **120**.

[0199] Among the instructions shown in Figure **14**, Instructions 3 and 5 are composed of four bytes, Instructions 1, 2, 4, 6 and 7 are composed of three bytes and Instruction 8 is composed of one byte. That is to say, the total code size of the instructions shown in Figure **14** is 24 bytes.

[0200] Figure **15** illustrates respective instructions obtained by rewriting some of the instructions shown in Figure **14** into associated instructions in the third instruction format. Specifically, Instructions 4 and 5 shown in Figure **14** are merged into rewritten Instruction 4 **DMACH_SUB** and Instructions 7 and 8 shown in Figure **14** are merged into rewritten Instruction 6 **MOV_LGE**.

[0201] In Figure **15**, Instruction 1 represents a half-word transfer instruction from a register to a memory in the first instruction format. Instruction 2 represents a half-word swap instruction (swap) in the second instruction format. Instruction 3 represents a half-word transfer instruction from a register to a memory in the second instruction format. Instruction 4 represents an instruction that a dual half-word multiply-and-accumulate operation and subtraction of an immediate value should be executed in parallel to each other. Instruction 5 represents a memory-to-register transfer instruction in the second instruction format. And Instruction 6 represents an instruction that memory-to-register data transfer and conditional branch should be executed in parallel to each other.

[0202] Hereinafter, specific times of operations will be described.

[0203] Instruction 1 MOVH **D0, (#-6, A3)** is a register-to-memory transfer instruction (store) in the first instruction format that a value in the data register D0 should be stored at a memory address obtained by subtracting "6" from an address stored in the address register **A3**.

(a) IF stage

[0204] The instruction fetch section **102** reads out the instruction from the instruction memory **101** and then stores the instruction in the instruction register **103**.

(b) DEC stage

[0205] The instruction-type identifier **111** decodes the instruction stored in the instruction register **103** to identify the type of this instruction as the first instruction format and output a control signal indicative of the type to the first register address selector **117**. The first register address extractor **112** extracts source register addresses "00" and "11", which are the respective addresses of the data register **D0** and the address register **A3**. The register address converter **113** converts the address "00" of the data register **D0** and the address "11" of the address register **A3** into 4-bit addresses "1100" and "1011", respectively.

[0206] Responsive to the control signal supplied from the instruction-type identifier **111**, the first register address selector **117** outputs the respective 4-bit addresses "1100" and "1011" of the data and address registers **D0** and **A3**, supplied from the register address converter **113**, to the register file **120**.

[0207] The operation decoder **116** decodes the instruction as a register-to-memory transfer instruction (store) that a value in the data register **D0** should be stored at a memory address obtained by subtracting "6" from an address stored in the address register **A3**, thereby producing an associated control signal.

[0208] The contents of the data register **D0**, associated with the physical address "1100", are read out as a source operand from the register file **120** and then stored in the E0 register **131**. And the contents of the address register **A3**, associated with the physical address "1011", are read out as another source operand from the register file **120** and then stored in the E1 register **132**.

(c) EX stage

[0209] The data stored in the E0 register **131** is passed through the operation unit **133** and then stored in the M0 register **141**. In response to the control signal supplied from the operation decoder **116**, the operation unit **133** subtracts "6" from the address stored in the E1 register **132** and outputs the result of subtraction to an address input section of the data memory **143**.

(d) MEM stage

[0210] The data stored in the M0 register **141** is stored in the data memory **143** at an address specified by the result of subtraction performed by the operation unit **133**.

(e) WB stage

[0211] No operations are performed.

[0212] Instruction 2 **SWHW E0, E2** is an instruction in the second instruction format that data, obtained by swapping the upper and lower half words of the data stored in the extended register **E0**, should be stored in the extended register **E2**.

(a) IF stage

[0213] The instruction fetch section **102** reads out the instruction from the instruction memory **101** and then stores the instruction in the instruction register **103**.

(b) DEC stage

[0214] The instruction-type identifier **111** decodes the instruction stored in the instruction register **103** to identify the type of this instruction as the second instruction format and output a control signal indicative of the type to the first register address selector **117**. The second register address extractor **114** extracts a source register address "0000", which is the address of the extended register **E0**, and a destination register address "0010", which is the address of the extended register **E2**.

[0215] Responsive to the control signal supplied from the instruction-type identifier **111**, the first register address selector **117** outputs the addresses, supplied from the second register address extractor **114**, to the register file **120**. The operation decoder **116** decodes the instruction to find that data, obtained by swapping the upper and lower half words of the data stored in the extended register **E0**, should be stored in the extended register **E2**, thereby producing an associated control signal.

[0216] The contents of the extended register **E0**, associated with the physical address "0000", are read out as a source operand from the register file **120** and then stored in the E0 register **131**.

(c) EX stage

[0217] Responsive to the control signal supplied from the operation decoder **116**, the operation unit **133** performs a swapping operation on the data stored in the E0 register **131** by swapping the upper and lower half words thereof. Then, the result of this operation is stored in the M0 register **141**.

(d) MEM stage

[0218] The data stored in the M0 register **141** is passed through the data memory **143** and then stored in the W0 register **151**.

(e) WB stage

[0219] The data stored in the W0 register **151** is read out and then stored in the register file **120** at the destination address "0010" specified by the second register address extractor **114**.

[0220] Instruction 3 **MOVH E2, (#-4, A3)** is a register-to-memory transfer instruction (store) in the second instruction format that a value in the extended register **E2** should be stored at a memory address obtained by subtracting "4" from an address stored in the address register **A3**.

(a) IF stage

[0221] The instruction fetch section **102** reads out the instruction from the instruction memory **101** and then stores the instruction in the instruction register **103**.

(b) DEC stage

[0222] The instruction-type identifier **111** decodes the instruction stored in the instruction register **103** to identify the type of this instruction as the second instruction format and output a control signal indicative of the type to the first register address selector **117**. The first register address extractor **112** extracts a source register address "11", which is the address of the address register **A3**. The second register address extractor **114** extracts another source register address "0010", which is the address of the extended register **E2**. The register address converter **113** converts the address "11" of the address register **A3** into a 4-bit address "1011".

[0223] Responsive to the control signal supplied from the instruction-type identifier **111**, the first register address selector **117** outputs the 4-bit address "1011" of the address register **A3**, supplied from the register address converter **113**, and the 4-bit address "0010" of the extended register **E2** to the register file **120**.

[0224] The operation decoder **116** decodes the instruction as a register-to-memory transfer instruction (store) that a value in the extended register **E2** should be stored at a memory address obtained by subtracting "4" from an address stored in the address register **A3**, thereby producing an associated control signal.

[0225] The contents of the extended register **E2**, associated with the physical address "0010", are read out as a source operand from the register file **120** and then stored in the E0 register **131**. And the contents of the address register **A3**, associated with the physical address "1011", are read out as another source operand from the register file **120** and then stored in the E1

register **132**.

(c) EX stage

[0226] The data stored in the E0 register **131** is passed through the operation unit **133** and then stored in the M0 register **141**. In response to the control signal supplied from the operation decoder **116**, the operation unit **133** subtracts "4" from the address stored in the E1 register **132** and outputs the result of subtraction to an address input section of the data memory **143**.

(d) MEM stage

[0227] The data stored in the M0 register **141** is stored in the data memory **143** at an address specified by the result of subtraction performed by the operation unit **133**.

(e) WB stage

[0228] No operations are performed.

[0229] Instruction 4 **DMACH_SUB E6, E1, 1, E3** is an instruction in the third instruction format, and is composed of two units to be executed in parallel to each other. One of these units is an instruction that a product obtained by multiplying together respective upper half words of the data stored in the extended registers **E6** and **E1** should be added to a product obtained by multiplying together respective lower half words thereof, and the sum should be accumulated in the extended register **E1**. The other unit is an instruction that "1" should be subtracted from the data stored in the extended register **E3**.

(a) IF stage

[0230] The instruction fetch section **102** reads out the instruction from the instruction memory **101** and then stores the instruction in the instruction register **103**.

(b) DEC stage

[0231] The instruction-type identifier **111** decodes the instruction stored in the instruction register **103** to identify the type of this instruction as the third instruction format and output a control signal indicative of the type to the first register address selector **117**. The second register address extractor **114** extracts a source register address "0110", which is the address of the extended register **E6**, and a destination register address "0001", which is the address of the extended register **E1**. In addition, the third register address extractor **115** extracts "0011" as a source/destination register address, i.e., the address of the extended register E3.

[0232] Responsive to the control signal supplied from the instruction-type identifier **111**, the first register address selector **117** outputs the addresses, supplied from the second register address extractor **114**, to the register file **120**. The operation decoder **116** decodes the instruction to find that a product obtained by multiplying together respective upper half words of the data stored in the extended registers **E6** and **E1** should be added to a product obtained by multiplying together respective lower half words thereof, and the sum should be accumulated in the extended register **E1**, thereby producing an associated control signal.

[0233] The contents of the extended register **E6**, associated with the physical address "0110", are read out as a source operand from the register file **120** and then stored in the E0 register **131**. And the contents of the extended register **E1**, associated with the physical address "0001", are read out as another source operand from the register file **120** and then stored in the E1 register **132**. In parallel to these operations, the contents of the extended register **E3**, associated with the physical address "0011", are read out as still another source operand from the register file **120** and then stored in the E2 register **134**.

(c) EX stage

[0234] The operation unit **133** extracts and multiplies together the respective upper half words of the data stored in the E0 and E1 registers **131** and **132**. At the same time, the operation unit **133** extracts and multiplies together the respective lower half words of the data stored in the E0 and E1 registers **131** and **132**. These two products are added to the data stored in the E1 register **132**. And the sum is stored in the M0 register **141**. In parallel thereto, the operation unit **133** also reads out the data stored in the E2 register **134** and subtracts "1" therefrom. And the result of subtraction is stored in the M1 register **142**.

(d) MEM stage

[0235] The data stored in the M0 register **141** is passed through the data memory **143** and then stored in the W0 register **151**. Also, the data stored in the M1 register **142** is passed through the data memory **143** and then stored in the W1 register **152**.

(e) WB stage

[0236] The data stored in the W0 register **151** is read out and then stored in the register file **120** at the destination address "0001" specified by the second register address extractor **114**. In parallel thereto, the data stored in the W1 register **152** is read out and then stored in the register file **120** at the destination address "0011" specified by the third register address extractor **115**.

[0237] Instruction 5 **MOV (#4, E4+), E6** is a post-increment memory-to-register transfer instruction in the second instruction format that data should be read out from a memory address of the extended register **E4** and then

stored in the extended register **E6** and the value of the extended register **E4** should be increased by "4" after the storage.

(a) IF stage

[0238] The instruction fetch section **102** reads out the instruction from the instruction memory **101** and then stores the instruction in the instruction register **103**.

(b) DEC stage

[0239] The instruction-type identifier **111** decodes the instruction stored in the instruction register **103** to identify the type of this instruction as the second instruction format and output a control signal indicative of the type to the first register address selector **117**. The second register address extractor **114** extracts a source register address "0100", which is the address of the extended register **E4**, and a destination register address "0110", which is the address of the extended register **E6**.

[0240] Responsive to the control signal supplied from the instruction-type identifier **111**, the first register address selector **117** outputs the addresses, supplied from the second register address extractor **114**, to the register file **120**. The operation decoder **116** decodes the instruction as a post-increment memory-to-register transfer instruction that data should be read out from a memory address of the extended register **E4** and then stored in the extended register **E6** and the value of the extended register **E4** should be increased by "4" after the storage, thereby producing an associated control signal.

[0241] The contents of the extended register **E4**, associated with the physical address "0100", are read out as a source operand from the register file **120** and then stored in the E0 register **131**.

(c) EX stage

[0242] The operation unit **133** reads out the data stored in the E0 register **131** and outputs the data to an address input section of the data memory **143** in response to the control signal supplied from the operation decoder **116**. Also, the operation unit **133** adds "4" to the data read out, and stores the sum in the M0 register **141**.

(d) MEM stage

[0243] The data stored in the M0 register **141** is passed through the data memory **143** and then stored in the W0 register **151**. Also, responsive to the control signal supplied from the operation decoder **116**, data is read out from the data memory **143** at the specified memory address and then stored in the W0 register **151**.

(e) WB stage

[0244] The data stored in the W0 register **151** is read out and then stored in the register file **120** at the destination address "0110" specified by the second register address extractor **114**.

[0245] Instruction 6 **MOV_LGE (#4, E5+), E1** is an instruction in the third instruction format and is composed of two units to be executed in parallel to each other. One of these two units is a post-increment memory-to-register transfer instruction that data should be read out from a memory address of the extended register **E5** and then stored in the extended register **E1** and the value of the extended register **E5** should be increased by "4" after the storage. The other unit is an instruction that if the result of subtraction performed to execute Instruction 4 is equal to or larger than "0", then a loop instruction should be executed by changing the value of the program counter **124** into an address **LOOP** shown in Figure **15** above Instruction **1**.

(a) IF stage

[0246] The instruction fetch section **102** reads out the instruction from the instruction memory **101** and then stores the instruction in the instruction register **103**.

(b) DEC stage

[0247] The instruction-type identifier **111** decodes the instruction stored in the instruction register **103** to identify the type of this instruction as the third instruction format and output a control signal indicative of the type to the first register address selector **117**. The second register address extractor **114** extracts a source register address "0101", which is the address of the extended register **E5**, and a destination register address "0001", which is the address of the extended register **E1**.

[0248] Responsive to the control signal supplied from the instruction-type identifier **111**, the first register address selector **117** outputs the addresses, supplied from the second register address extractor **114**, to the register file **120**. The operation decoder **116** decodes the instruction as a post-increment memory-to-register transfer instruction that data should be read out from a memory address of the extended register **E5** and then stored in the extended register **E1** and the value of the extended register **E5** should be increased by "4" after the storage, thereby producing an associated control signal.

[0249] The contents of the extended register **E5**, associated with the physical address "0101", are read out as a source operand from the register file **120** and then stored in the E0 register **131**.

[0250] Furthermore, the operation decoder **116** decodes the other unit as a loop instruction, thereby producing an associated control signal.

[0251] Responsive to the control signal supplied from

the operation decoder **116**, the value of the program counter **124** in the register file **120** is stored in the E2 register **134**.

(c) EX stage

[0252] The operation unit **133** reads out the data stored in the E0 register **131** and outputs the data to an address input section of the data memory **143** in response to the control signal supplied from the operation decoder **116**. Also, the operation unit **133** adds "4" to the data read out, and stores the sum in the M0 register **141**.

[0253] On the other hand, the operation unit **136** reads out the data stored in the E2 register **134** and stores the destination address of the loop in the M1 register **142** responsive to the control signal supplied from the operation decoder **116**.

(d) MEM stage

[0254] The data stored in the M0 register **141** is passed through the data memory **143** and then stored in the W0 register **151**. Also, responsive to the control signal supplied from the operation decoder **116**, data is read out from the data memory **143** at the specified memory address and then stored in the W0 register **151**.

[0255] Also, the data stored in the M1 register **142** is passed through the data memory **143** and then stored in the W1 register **152**.

(e) WB stage

[0256] The data stored in the W0 register **151** is read out and then stored in the register file **120** at the destination address "0001" specified by the second register address extractor **114**.

[0257] On the other hand, the data stored in the W1 register **152** is stored at the program counter **124** in the register file **120**.

[0258] Among the instructions shown in Figure **15**, Instructions 3, 4, 5 and 6 are composed of four bytes, while Instructions 1 and 2 are composed of three bytes. That is to say, the total code size of the instructions shown in Figure 15 is 22 bytes.

[0259] As can be understood, by executing some instructions in the third instruction format according to this embodiment, the code size can be reduced from 24 to 22 bytes. Also, by changing the instruction set shown in Figure **14** into that shown in Figure **15**, the number of instructions can also be reduced from eight to six, thus improving the performance in executing the instructions.

[0260] It should be noted that the present invention is in no way limited to the data processor described in the foregoing embodiment. For example, although address and data registers are extended in the foregoing embodiment, general-purpose registers may be extended instead. Also, in the foregoing description, the respective numbers of address, data and extended registers are four, four and eight. Alternatively, any number of registers may be used for each of these types. Moreover, in the foregoing embodiment, the first register file **220** is specified in the first instruction format, while the second register file **120**, including the first register file **220**, is specified in the second instruction format. However, the present invention is not limited to such a specific embodiment. The number of registers included in the second register file may be larger than that of registers included in the first register file, and only the second register file may be specified in the second instruction format without specifying the first register file.

[0261] Furthermore, in accordance with the third instruction format, two units are supposed to be executed in parallel to each other in the foregoing embodiment. If necessary, it is naturally possible to execute three or more units in parallel to each other in accordance with the third instruction format.

Claims

1. A data processor for executing an instruction described in a first instruction format and an instruction described in a second instruction format, wherein the first instruction format defines a register-addressing field of a predetermined size, while the second instruction format defines a register-addressing field of a size larger than the size of the register-addressing field defined by the first instruction format, and
   wherein the data processor comprises:

   means, responsive to an instruction, for identifying the received instruction as being described in the first or second instruction format by the instruction itself;
   a first register file including a plurality of registers; and
   a second register file also including a plurality of registers, the number of the registers included in the second register file being larger than the number of the registers included in the first register file,
   wherein if the identifying means has identified the received instruction as being described in the first instruction format, the data processor executes the instruction using data held in the first register file,
   while if the identifying means has identified the received instruction as being described in the second instruction format, the data processor executes the instruction using data held in the second register file.

2. The data processor of Claim 1, wherein the first instruction format defines a number of instruction

fields and the second instruction format defines another number of instruction fields, and
wherein the identifying means identifies the received instruction as being described in the first or second instruction format by the contents of at least one of the instruction fields of the instruction that is defined by at least one predetermined ordinal number.

3. The data processor of Claim 2, wherein the number of the instruction fields defined by the second instruction format is larger than the number of the instruction fields defined by the first instruction format.

4. The data processor of Claim 2, wherein the predetermined ordinal number of the instruction field used by the identifying means for format identification is first.

5. The data processor of Claim 2, wherein the predetermined ordinal numbers of the respective instruction fields used by the identifying means for format identification are first and second.

6. The data processor of Claim 2, wherein the instructions described in the first and second instruction formats are variable-length instructions.

7. The data processor of Claim 2, wherein in the first instruction format, the register-addressing field is included in the instruction field that is defined by the predetermined ordinal number and used by the identifying means for format identification, and
wherein in the second instruction format, the register-addressing field is not included in the instruction field that is defined by the predetermined ordinal number and used by the identifying means for format identification.

8. The data processor of Claim 2, wherein in the first instruction format, operation and length of an instruction to be executed or only the length of the instruction are/is specified in a first one of the instruction fields, and only operations of the instruction to be executed are specified in succeeding ones of the instruction fields.

9. The data processor of Claim 2, wherein in the second instruction format, only length of an instruction to be executed is specified in a first one of the instruction fields, and only operations of the instruction to be executed are specified in succeeding ones of the instruction fields.

10. The data processor of Claim 1 or 2, wherein the second register file includes all of the registers included in the first register file.

11. The data processor of Claim 1 or 2, wherein an address described in the first instruction format for specifying one of the registers included in the first register file is different from an address described in the second instruction format for specifying the same register.

12. The data processor of Claim 11, further comprising an address converter,
wherein responsive to an instruction, the address converter converts an address described in the first instruction format, which has been specified by the instruction to access one of the registers included in the first register file, into an address of the register described in the second instruction format, and
wherein if the identifying means has identified the received instruction as being described in the first instruction format, then the address, converted by the address converter, is output to the first register file.

13. The data processor of Claim 2, wherein the first instruction format defines an instruction field specifying operation of the instruction and an address of one of the registers included in the first register file.

14. The data processor of Claim 2, wherein the second instruction format defines an instruction field specifying only operation of the instruction and another instruction field specifying only an address of at least one of the registers included in the first and second register files.

15. The data processor of Claim 1 or 2, further comprising a plurality of operation units for executing arithmetic operations specified by the instructions in parallel using the data held in the first and second register files.

16. The data processor of Claim 1 or 2, further executing an instruction described in a third instruction format,
wherein the third instruction format specifies a plurality of operations and defines a register-addressing field of a size larger than that of the register-addressing field defined by the first instruction format, the register-addressing field defined by the third instruction format being used to specify one of the registers included in the second register file, and
wherein responsive to an instruction, the identifying means identifies the received instruction as being described in the third instruction format by the instruction itself.

17. The data processor of Claim 16, wherein the third instruction format defines: an instruction field specifying a plurality of operations to be executed follow-

ing an instruction; and another instruction field addressing one of the registers included in the second register file.

18. The data processor of Claim 16, further comprising a plurality of operation units for executing in parallel the operations specified by the instruction described in the third instruction format.

19. The data processor of Claim 16, wherein an instruction described in the third instruction format is an instruction described in the second instruction format that is frequently specified.

20. The data processor of Claim 16, wherein the number of bits of each of operation-specifying fields, which are defined by the third instruction format for specifying operations of an instruction to be executed in parallel to each other, is smaller than the number of bits of an operation-specifying field defined by the second instruction format.

21. The data processor of Claim 16, wherein an address, described in the second instruction format for specifying one of the registers included in the second register file, is the same as an address described in the third instruction format for specifying the same register.

22. A data processor for executing an instruction described in a first instruction format and an instruction described in a second instruction format, the data processor comprising:

a register file including a predetermined number of registers, an address described in the first instruction format for specifying one of the registers being different from an address described in the second instruction format for specifying the same register;
an address converter for receiving the instruction described in the first instruction format and converting an address described in the first instruction format, specified by the instruction to access one of the registers, into an address described in the second instruction format; and
means, responsive to an instruction, for identifying the received instruction as being described in the first or second instruction format by the instruction itself,
wherein the output of the address converter is controlled by the output of the identifying means.

23. An instruction format for use in defining an arrangement of an instruction to be executed by a data processor, the instruction format being implemented as first and second instruction formats,
wherein the first instruction format defines a number of instruction fields and the second instruction format defines another number of instruction fields, the number of the instruction fields defined by the second instruction format being larger than the number of the instruction fields defined by the first instruction format, and
wherein at least one of the instruction fields that are defined by the first and second instruction formats is used to identify the type of the instruction to be executed as being described in the first or second instruction format, and
wherein the first instruction format defines a register-addressing field of a predetermined size, while the second instruction format defines a register-addressing field of a size larger than the size of the register-addressing field defined by the first instruction format.

24. The instruction format of Claim 23, implemented as a third instruction format,
wherein the third instruction format defines still another number of instruction fields, the number of the instruction fields defined by the third instruction format being larger than the number of the instruction fields defined by the first instruction format, and
wherein the third instruction format defines a register-addressing field of a size larger than the size of the register-addressing field defined by the first instruction format, and
wherein at least one of the instruction fields that is defined by the third instruction format is used to identify the type of the instruction to be executed as being described in the third instruction format, and
wherein the third instruction format describes a plurality of operations to be executed.

25. The instruction format of Claim 23 or 24, wherein the instruction field used to identify the type of the instruction to be executed is a first one of the instruction fields.

26. The instruction format of Claim 23 or 24, wherein the instruction fields used to identify the type of the instruction to be executed are first and second ones of the instruction fields.

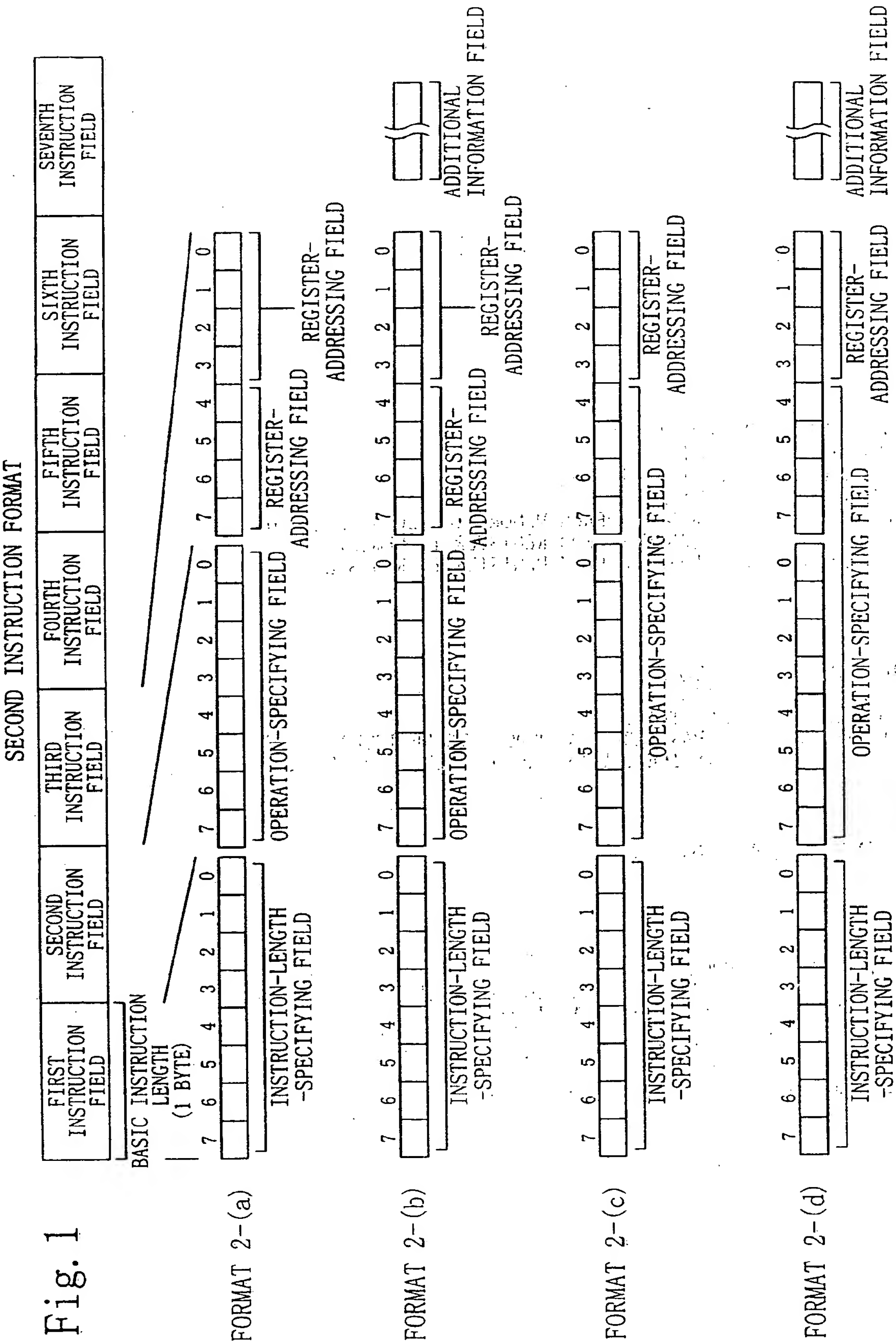

SECOND INSTRUCTION FORMAT
FIRST INSTRUCTION FIELD
SECOND INSTRUCTION FIELD
THIRD INSTRUCTION FIELD
FOURTH INSTRUCTION FIELD
FIFTH INSTRUCTION FIELD
SIXTH INSTRUCTION FIELD
SEVENTH INSTRUCTION FIELD
BASIC INSTRUCTION LENGTH (1 BYTE)
FORMAT 2-(a)
FORMAT 2-(b)
FORMAT 2-(c)
FORMAT 2-(d)
7 6 5 4 3 2 1 0
INSTRUCTION-LENGTH -SPECIFYING FIELD
OPERATION-SPECIFYING FIELD
REGISTER-ADDRESSING FIELD
ADDITIONAL INFORMATION FIELD

Fig. 1

Fig. 2

```
SECOND INSTRUCTION FORMAT (a)
  ADD Rm, Rn          : ADD
  SUB Rm, Rn          : SUBTRACT
  CMP Rm, Rn          : COMPARE
  MOV(Rm), Rn         : TRANSFER FROM MEMORY TO REGISTER (LOAD)
  MOV Rm, (Rn)        : TRANSFER FROM REGISTER TO MEMORY (STORE)
  MOV Rm, Rn          : TRANSFER FROM REGISTER TO REGISTER
  ...

SECOND INSTRUCTION FORMAT (b)
  ADD Rm, Rn, Rd      : ADD
  SUB Rm, Rn, Rd      : SUBTRACT
  MOV(Ri, Rm), Rn     : TRANSFER FROM MEMORY TO REGISTER (LOAD)
  ...                   INDIRECTLY BY WAY OF INDEXED REGISTER

SECOND INSTRUCTION FORMAT (c)
  ...

SECOND INSTRUCTION FORMAT (d)
  ADD imm 16, Rn      : ADD 16-BIT IMMEDIATE VALUE
  ADD imm 16, Rn      : ADD 16-BIT IMMEDIATE VALUE
  MOV(disp8, SP), Rn  : TRANSFER FROM MEMORY TO REGISTER (LOAD)
                        BY ADDRESSING USING STACK POINTER (SP) WITH DISPLACEMENT
  MOV Rm, (disp8, SP) : TRANSFER FROM REGISTER TO MEMORY (STORE)
  ...                   BY ADDRESSING USING STACK POINTER (SP) WITH DISPLACEMENT
```

Fig. 3

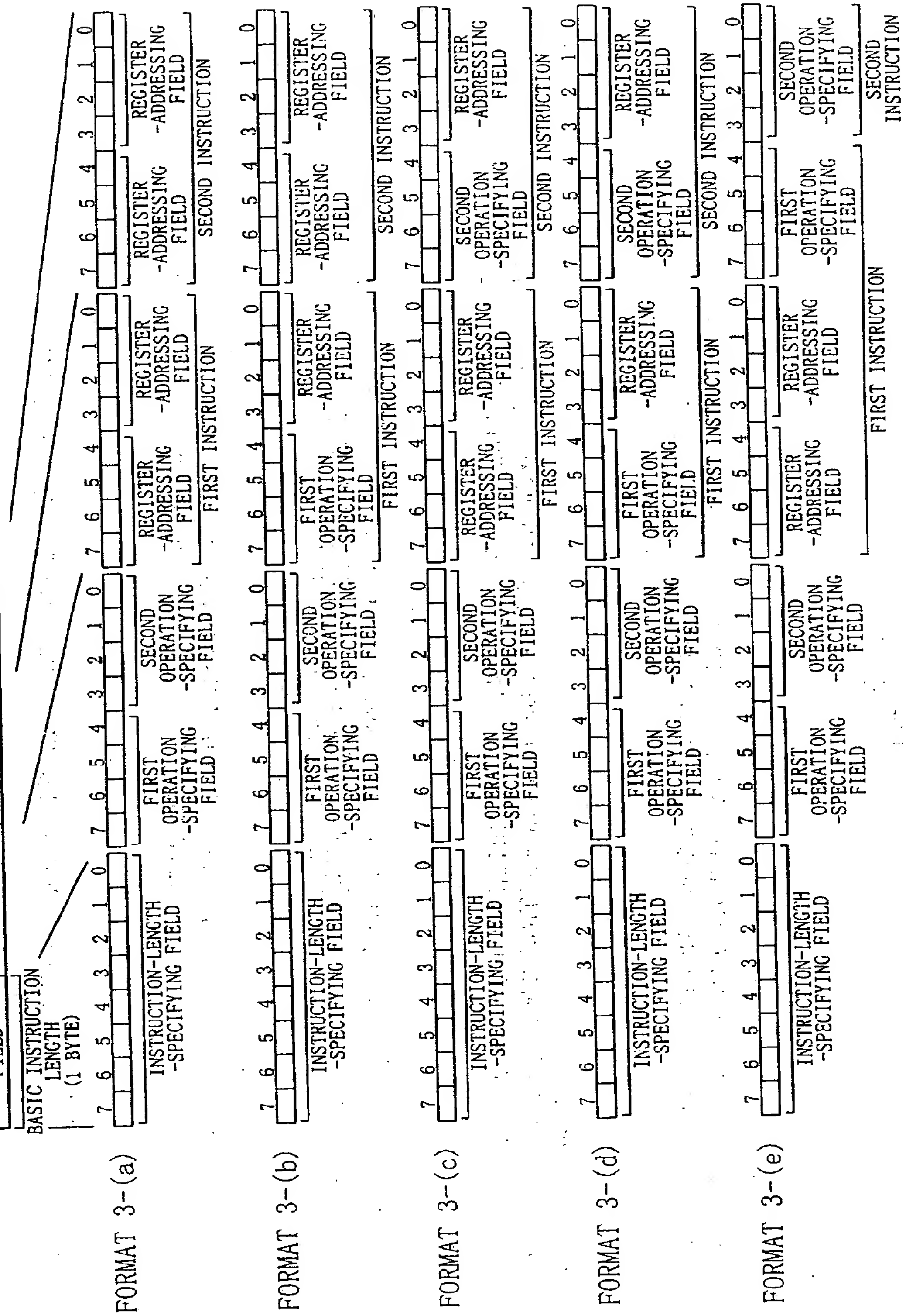

THIRD INSTRUCTION FORMAT
FIRST INSTRUCTION FIELD
SECOND INSTRUCTION FIELD
THIRD INSTRUCTION FIELD
FOURTH INSTRUCTION FIELD
BASIC INSTRUCTION LENGTH (1 BYTE)
FORMAT 3-(a)
FORMAT 3-(b)
FORMAT 3-(c)
FORMAT 3-(d)
FORMAT 3-(e)
INSTRUCTION-LENGTH -SPECIFYING FIELD
FIRST OPERATION -SPECIFYING FIELD
SECOND OPERATION -SPECIFYING FIELD
REGISTER -ADDRESSING FIELD
FIRST INSTRUCTION
SECOND INSTRUCTION

Fig. 4

```
THIRD INSTRUCTION FORMAT (a)
  ADD_ADD Rm1, Rn1, Rm2, Rn2      : ADD BETWEEN REGISTERS AND ADD BETWEEN REGISTERS IN PARALLEL
  SUB_ASL Rm1, Rn1, Rm2, Rn2      : SUBTRACT BETWEEN REGISTERS AND SHIFT BETWEEN REGISTERS IN PARALLEL
  ...

THIRD INSTRUCTION FORMAT (b)
  ADD_ADD imm4, Rn1, Rm2, Rn2     : ADD IMMEDIATE VALUE TO REGISTER AND ADD BETWEEN REGISTERS IN PARALLEL
  SUB_ASL imm4, Rn1, Rm2, Rn2     : SUBTRACT IMMEDIATE VALUE FROM REGISTER AND SHIFT BETWEEN REGISTERS IN PARALLEL
  ...

THIRD INSTRUCTION FORMAT (c)
  ADD_ADD Rm1, Rn1, imm4, Rn2     : ADD BETWEEN REGISTERS AND ADD IMMEDIATE VALUE TO REGISTER IN PARALLEL
  DMACH_SUB Rm1, Rn1, imm4, Rn2   : MULTIPLY AND ACCUMULATE HALF-WORD BETWEEN REGISTERS AND
                                    SUBTRACT IMMEDIATE VALUE FROM REGISTER IN PARALLEL
  ...

THIRD INSTRUCTION FORMAT (d)
  ADD_ADD imm4, Rn1, imm4, Rn2    : ADD IMMEDIATE VALUE TO REGISTER AND ADD IMMEDIATE VALUE TO REGISTER IN PARALLEL
  SUB_ADD imm4, Rn1, imm4, Rn2    : SUBTRACT IMMEDIATE VALUE FROM REGISTER AND ADD IMMEDIATE VALUE TO REGISTER IN PARALLEL
  ...

THIRD INSTRUCTION FORMAT (e)
  MOV_Lcc(Rm+, imm4), Rn          : TRANSFER BETWEEN REGISTERS AND CONDITIONAL LOOP IN PARALLEL
  ...
```

Fig. 5

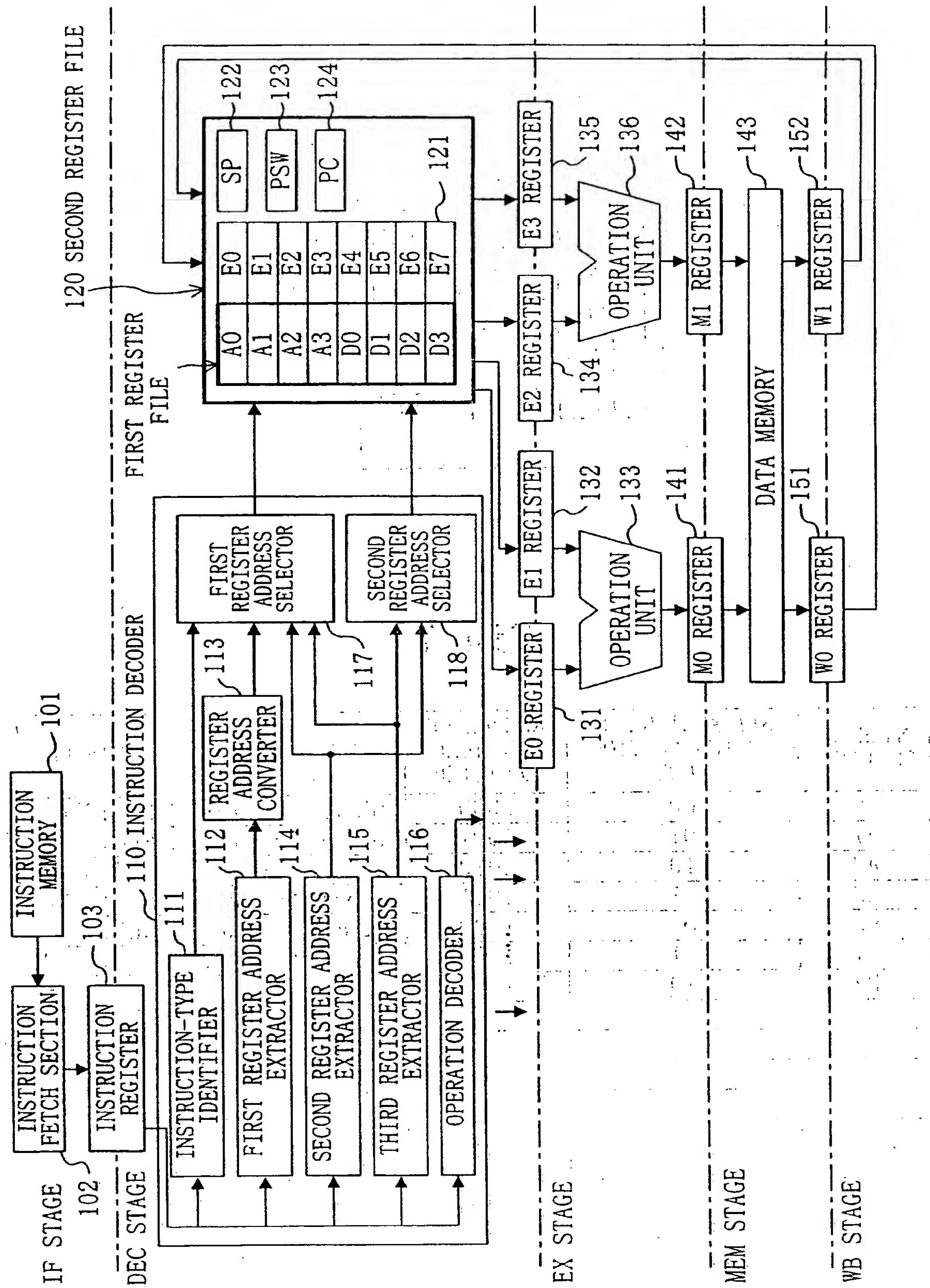

IF STAGE
DEC STAGE
EX STAGE
MEM STAGE
WB STAGE
INSTRUCTION MEMORY
101
INSTRUCTION FETCH SECTION
102
INSTRUCTION REGISTER
103
110 INSTRUCTION DECODER
111
INSTRUCTION-TYPE IDENTIFIER
112
FIRST REGISTER ADDRESS EXTRACTOR
113
REGISTER ADDRESS CONVERTER
114
SECOND REGISTER ADDRESS EXTRACTOR
115
THIRD REGISTER ADDRESS EXTRACTOR
116
OPERATION DECODER
117
FIRST REGISTER ADDRESS SELECTOR
118
SECOND REGISTER ADDRESS SELECTOR
FIRST REGISTER FILE
120 SECOND REGISTER FILE
121
A0
A1
A2
A3
D0
D1
D2
D3
E0
E1
E2
E3
E4
E5
E6
E7
SP
122
PSW
123
PC
124
E0 REGISTER
131
E1 REGISTER
132
OPERATION UNIT
133
E2 REGISTER
134
E3 REGISTER
135
OPERATION UNIT
136
M0 REGISTER
141
M1 REGISTER
142
DATA MEMORY
143
W0 REGISTER
151
W1 REGISTER
152

Fig. 6

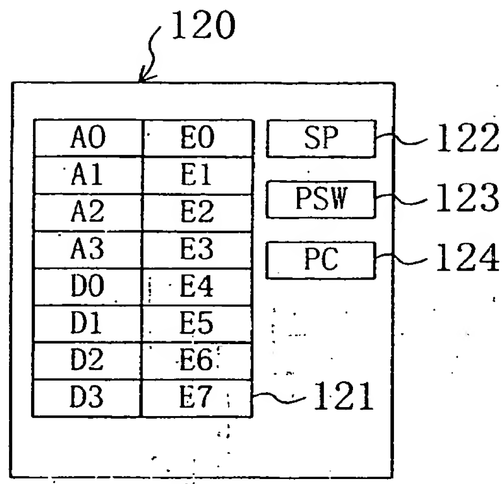


Fig. 7

NAME OF REGISTER	BIT ASSIGNMENT ON INSTRUCTION CODE	NUMBER OF PHYSICAL REGISTER	NAME OF PHYSICAL REGISTER
A0	00	1000	GENERAL-PURPOSE REGISTER
A1	01	1001	GENERAL-PURPOSE REGISTER
A2	10	1010	GENERAL-PURPOSE REGISTER
A3	11	1011	GENERAL-PURPOSE REGISTER
D0	00	1100	GENERAL-PURPOSE REGISTER
D1	01	1101	GENERAL-PURPOSE REGISTER
D2	10	1110	GENERAL-PURPOSE REGISTER
D3	11	1111	GENERAL-PURPOSE REGISTER
E0			
E1			
E2			
E3			
E4			
E5			
E6			
E7			

Fig. 8

NAME OF REGISTER	BIT ASSIGNMENT ON INSTRUCTION CODE	NUMBER OF PHYSICAL REGISTER	NAME OF PHYSICAL REGISTER
A0	1000	1000	GENERAL-PURPOSE REGISTER
A1	1001	1001	GENERAL-PURPOSE REGISTER
A2	1010	1010	GENERAL-PURPOSE REGISTER
A3	1011	1011	GENERAL-PURPOSE REGISTER
D0	1100	1100	GENERAL-PURPOSE REGISTER
D1	1101	1101	GENERAL-PURPOSE REGISTER
D2	1110	1110	GENERAL-PURPOSE REGISTER
D3	1111	1111	GENERAL-PURPOSE REGISTER
E0	0000	0000	GENERAL-PURPOSE REGISTER
E1	0001	0001	GENERAL-PURPOSE REGISTER
E2	0010	0010	GENERAL-PURPOSE REGISTER
E3	0011	0011	GENERAL-PURPOSE REGISTER
E4	0100	0100	GENERAL-PURPOSE REGISTER
E5	0101	0101	GENERAL-PURPOSE REGISTER
E6	0110	0110	GENERAL-PURPOSE REGISTER
E7	0111	0111	GENERAL-PURPOSE REGISTER

Fig. 9

...
a=a+b
c=c+d
...

Fig. 10

INSTRUCTION 1	ADD D1, D0
INSTRUCTION 2	ADD D2, D3

Fig. 11

INSTRUCTION 1	MOV (#4, SP), D0
INSTRUCTION 2	MOV (#8, SP), D1
INSTRUCTION 3	ADD D1, D0
INSTRUCTION 4	MOV D0, (#4, SP)
INSTRUCTION 5	MOV (#12, SP), D0
INSTRUCTION 6	MOV (#16, SP), D1
INSTRUCTION 7	ADD D1, D0
INSTRUCTION 8	MOV D0, (#12, SP)

Fig. 12

INSTRUCTION 1	MOV D0, (#20, SP)
INSTRUCTION 2	MOV D1, (#24, SP)
INSTRUCTION 3	MOV (#4, SP), D0
INSTRUCTION 4	MOV (#8, SP), D1
INSTRUCTION 5	ADD D1, D0
INSTRUCTION 6	MOV D0, (#4, SP)
INSTRUCTION 7	MOV (#12, SP), D0
INSTRUCTION 8	MOV (#16, SP), D1
INSTRUCTION 9	ADD D1, D0
INSTRUCTION 10	MOV D0, (#12, SP)
INSTRUCTION 11	MOV (#20, SP), D0
INSTRUCTION 12	MOV (#24, SP), D1

Fig. 13

INSTRUCTION 1	ADD E1, E0
INSTRUCTION 2	ADD E2, E3

Fig. 14

LOOP			CODE SIZE (BYTE)
INSTRUCTION 1	MOVH	D0, (#-6, A3)	3
INSTRUCTION 2	SWHW	E0, E2	3
INSTRUCTION 3	MOVH	E2, (#-4, A3)	4
INSTRUCTION 4	DMACH	E6, E1	3
INSTRUCTION 5	SUB	1, E3	4
INSTRUCTION 6	MOV	(#4, E4+), E6	3
INSTRUCTION 7	MOV	(#4, E5+), E1	3
INSTRUCTION 8	LGE		1

Fig. 15

LOOP			CODE SIZE (BYTE)
INSTRUCTION 1	MOVH	D0, (#-6, A3)	3
INSTRUCTION 2	SWHW	E0, E2	3
INSTRUCTION 3	MOVH	E2, (#-4, A3)	4
INSTRUCTION 4	DMACH_SUB	E6, E1, 1, E3	4
INSTRUCTION 5	MOV	(#4, E4+), E6	4
INSTRUCTION 6	MOV_LGE	(#4, E5+), E1	4

Fig. 16

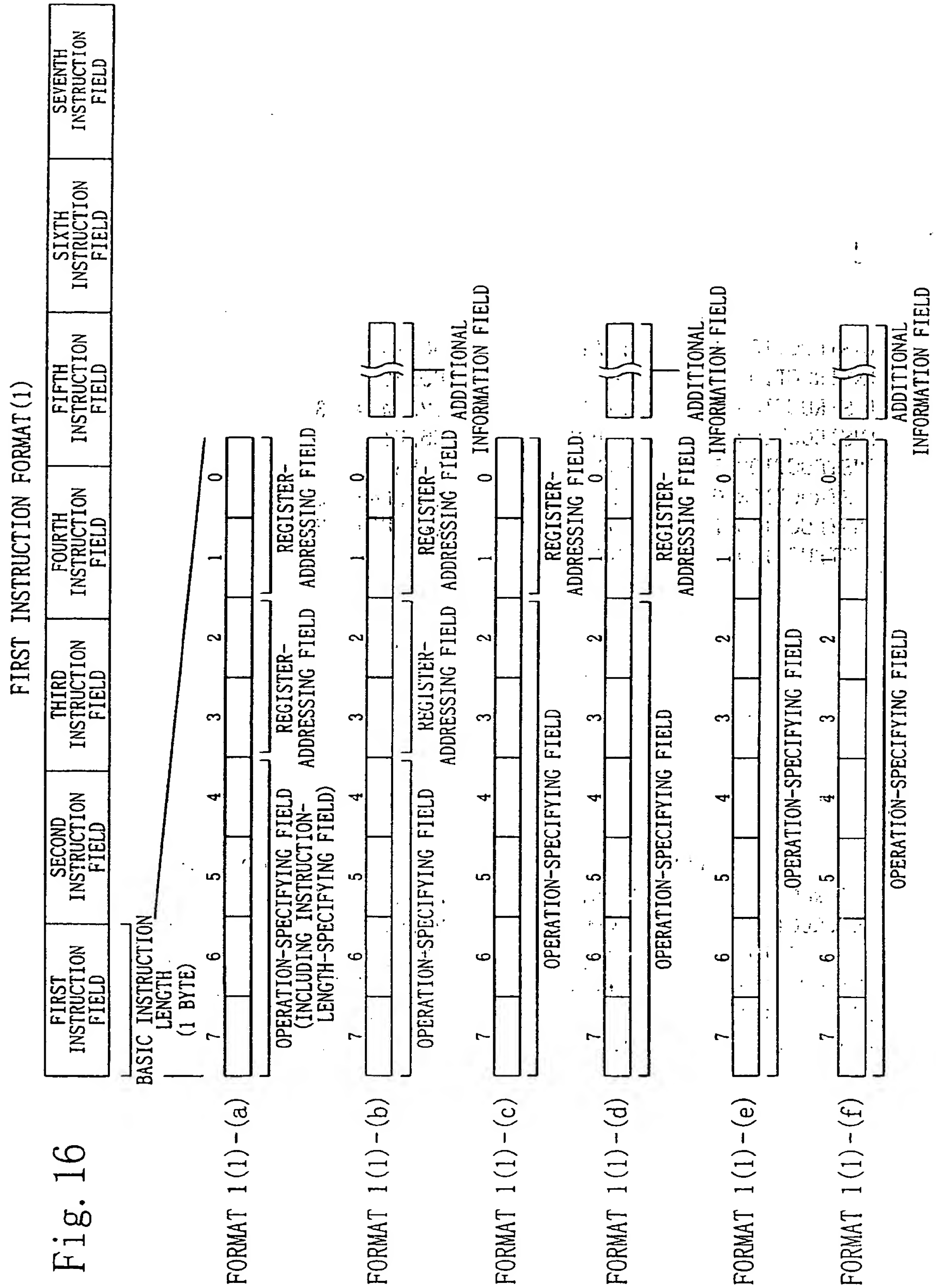

FIRST INSTRUCTION FORMAT (1)
FIRST INSTRUCTION FIELD
SECOND INSTRUCTION FIELD
THIRD INSTRUCTION FIELD
FOURTH INSTRUCTION FIELD
FIFTH INSTRUCTION FIELD
SIXTH INSTRUCTION FIELD
SEVENTH INSTRUCTION FIELD
BASIC INSTRUCTION LENGTH (1 BYTE)
FORMAT 1(1)-(a)
7 6 5 4 3 2 1 0
OPERATION-SPECIFYING FIELD (INCLUDING INSTRUCTION-LENGTH-SPECIFYING FIELD)
REGISTER-ADDRESSING FIELD
REGISTER-ADDRESSING FIELD
FORMAT 1(1)-(b)
7 6 5 4 3 2 1 0
OPERATION-SPECIFYING FIELD
REGISTER-ADDRESSING FIELD
REGISTER-ADDRESSING FIELD
ADDITIONAL INFORMATION FIELD
FORMAT 1(1)-(c)
7 6 5 4 3 2 1 0
OPERATION-SPECIFYING FIELD
REGISTER-ADDRESSING FIELD
FORMAT 1(1)-(d)
7 6 5 4 3 2 1 0
OPERATION-SPECIFYING FIELD
REGISTER-ADDRESSING FIELD
ADDITIONAL INFORMATION FIELD
FORMAT 1(1)-(e)
7 6 5 4 3 2 1 0
OPERATION-SPECIFYING FIELD
FORMAT 1(1)-(f)
7 6 5 4 3 2 1 0
OPERATION-SPECIFYING FIELD
ADDITIONAL INFORMATION FIELD

Fig. 17

```
FIRST INSTRUCTION FORMAT (1)-(a)
  ADD Dm,Dn           : ADD
  CMP Dm,Dn           : COMPARE
  CMP Am,An           : COMPARE
  MOV (Am),Dn         : TRANSFER FROM MEMORY TO REGISTER (LOAD)
  MOV Dm,(An)         : TRANSFER FROM REGISTER TO MEMORY (STORE)
  MOV Dm,Dn           : TRANSFER FROM REGISTER TO REGISTER
  MOV Am,An           : TRANSFER FROM REGISTER TO REGISTER
  ...

FIRST INSTRUCTION FORMAT (1)-(b)
  ...

FIRST INSTRUCTION FORMAT (1)-(c)
  CLR Dm              : CLEAR DATA TO 0
  INC Dm              : INCREASE DATA BY 1
  INC Am              : INCREASE DATA BY 1
  EXTB Dm             : EXTEND SIGNED BYTE DATA INTO WORD DATA
  EXTUB Dm            : EXTEND UNSIGNED BYTE DATA INTO WORD DATA
  EXTH Dm             : EXTEND SIGNED HALF-WORD DATA INTO WORD DATA
  EXTHU Dm            : EXTEND UNSIGNED HALF-WORD DATA INTO WORD DATA
  MOV SP,An           : TRANSFER STACK POINTER (SP) TO REGISTER
  ...

FIRST INSTRUCTION FORMAT (1)-(d)
  ADD imm8,An         : ADD 8-BIT IMMEDIATE VALUE
  ADD imm8,Dn         : ADD 8-BIT IMMEDIATE VALUE
  MOV imm16,An        : TRANSFER 16-BIT IMMEDIATE VALUE
  MOV imm16,Dn        : TRANSFER 16-BIT IMMEDIATE VALUE
  MOV (abs16),Dn      : TRANSFER FROM MEMORY TO REGISTER (LOAD) BY ADDRESSING WITH 16-BIT ABSOLUTE VALUE
  MOVBU (abs16),Dn    : TRANSFER ZERO-EXTENDED BYTE DATA FROM MEMORY TO REGISTER (LOAD)
                        BY ADDRESSING WITH 16-BIT ABSOLUTE VALUE
  MOVHU (abs16),Dn    : TRANSFER ZERO-EXTENDED HALF-WORD DATA FROM MEMORY TO REGISTER (LOAD)
                        BY ADDRESSING WITH 16-BIT ABSOLUTE VALUE
  MOV Dm,(abs16)      : TRANSFER FROM REGISTER TO MEMORY (STORE) BY ADDRESSING WITH 16-BIT ABSOLUTE VALUE
  MOVBU Dm,(abs16)    : TRANSFER ZERO-EXTENDED BYTE DATA FROM REGISTER TO MEMORY (STORE)
                        BY ADDRESSING WITH 16-BIT ABSOLUTE VALUE
  MOVHU Dm(abs16)     : TRANSFER ZERO-EXTENDED HALF-WORD DATA FROM REGISTER TO MEMORY (STORE)
                        BY ADDRESSING WITH 16-BIT ABSOLUTE VALUE
  MOV (disp8,SP),An   : TRANSFER FROM MEMORY TO REGISTER (LOAD) BY ADDRESSING USING STACK POINTER (SP) WITH DISPLACEMENT
  MOV (disp8,SP),Dn   : TRANSFER FROM MEMORY TO REGISTER (LOAD) BY ADDRESSING USING STACK POINTER (SP) WITH DISPLACEMENT
  MOV Am,(disp8,SP)   : TRANSFER FROM REGISTER TO MEMORY (STORE) BY ADDRESSING USING STACK POINTER (SP) WITH DISPLACEMENT
  MOV Dm,(disp8,SP)   : TRANSFER FROM REGISTER TO MEMORY (STORE) BY ADDRESSING USING STACK POINTER (SP) WITH DISPLACEMENT
  ...

FIRST INSTRUCTION FORMAT (1)-(e)
  NOP                 : NO OPERATION
  LOOP #CC            : LOOP CONTROL INSTRUCTION
  ...

FIRST INSTRUCTION FORMAT (1)-(f)
  BR #CC(disp8,PC)    : CONDITIONAL BRANCH OF PC WITH DISPLACEMENT
  JMP (disp8,PC)      : UNCONDITIONAL BRANCH OF PC WITH DISPLACEMENT
  ...
```

Fig. 18

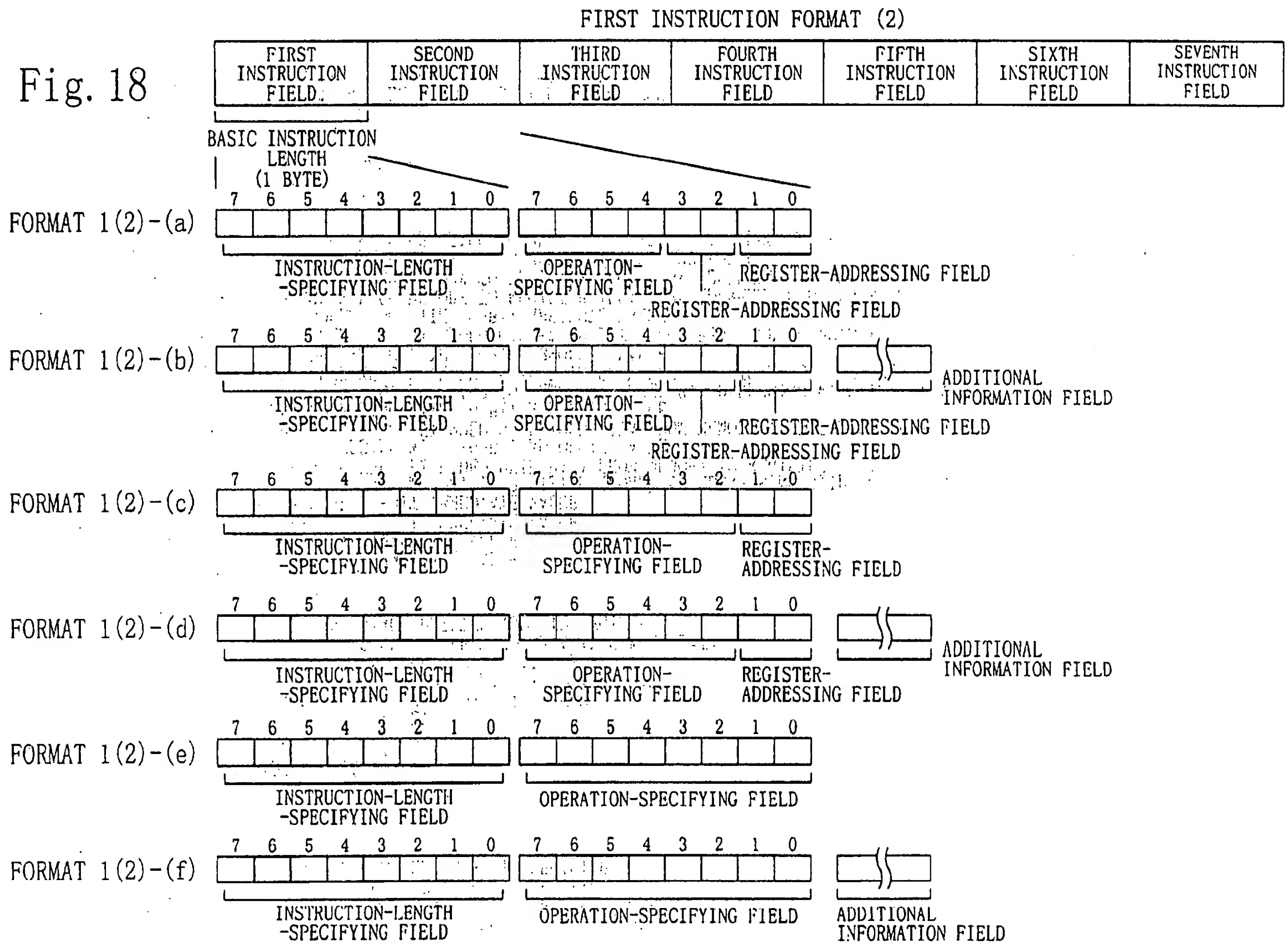

FIRST INSTRUCTION FORMAT (2)
FIRST INSTRUCTION FIELD
SECOND INSTRUCTION FIELD
THIRD INSTRUCTION FIELD
FOURTH INSTRUCTION FIELD
FIFTH INSTRUCTION FIELD
SIXTH INSTRUCTION FIELD
SEVENTH INSTRUCTION FIELD
BASIC INSTRUCTION LENGTH (1 BYTE)
FORMAT 1(2)-(a)
7 6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
INSTRUCTION-LENGTH -SPECIFYING FIELD
OPERATION-SPECIFYING FIELD
REGISTER-ADDRESSING FIELD
REGISTER-ADDRESSING FIELD
FORMAT 1(2)-(b)
7 6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
INSTRUCTION-LENGTH -SPECIFYING FIELD
OPERATION-SPECIFYING FIELD
REGISTER-ADDRESSING FIELD
REGISTER-ADDRESSING FIELD
ADDITIONAL INFORMATION FIELD
FORMAT 1(2)-(c)
7 6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
INSTRUCTION-LENGTH -SPECIFYING FIELD
OPERATION-SPECIFYING FIELD
REGISTER-ADDRESSING FIELD
FORMAT 1(2)-(d)
7 6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
INSTRUCTION-LENGTH -SPECIFYING FIELD
OPERATION-SPECIFYING FIELD
REGISTER-ADDRESSING FIELD
ADDITIONAL INFORMATION FIELD
FORMAT 1(2)-(e)
7 6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
INSTRUCTION-LENGTH -SPECIFYING FIELD
OPERATION-SPECIFYING FIELD
FORMAT 1(2)-(f)
7 6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
INSTRUCTION-LENGTH -SPECIFYING FIELD
OPERATION-SPECIFYING FIELD
ADDITIONAL INFORMATION FIELD

Fig. 19

```
FIRST INSTRUCTION FORMAT (2)-(a)
  SUB Dm,Dn            : SUBTRACT
  MOV(Am),An           : TRANSFER FROM MEMORY TO REGISTER (LOAD)
  MOV Am, (An)         : TRANSFER FROM REGISTER TO MEMORY (STORE)
  ...

FIRST INSTRUCTION FORMAT (2)-(b)
  MOV(Ai,Dn),Dn        : TRANSFER FROM MEMORY TO REGISTER (LOAD)
  ...                    INDIRECTLY BY WAY OF INDEXED REGISTER

FIRST INSTRUCTION FORMAT (2)-(c)
  ...

FIRST INSTRUCTION FORMAT (2)-(d)
  ADD imm16,An         : ADD 16-BIT IMMEDIATE VALUE
  ADD imm16,Dn         : ADD 16-BIT IMMEDIATE VALUE
  ...

FIRST INSTRUCTION FORMAT (2)-(e)
  RTI                  : RETURN FROM INTERRUPT STATE
  ...

FIRST INSTRUCTION FORMAT (2)-(f)
  ...
```

Fig. 20

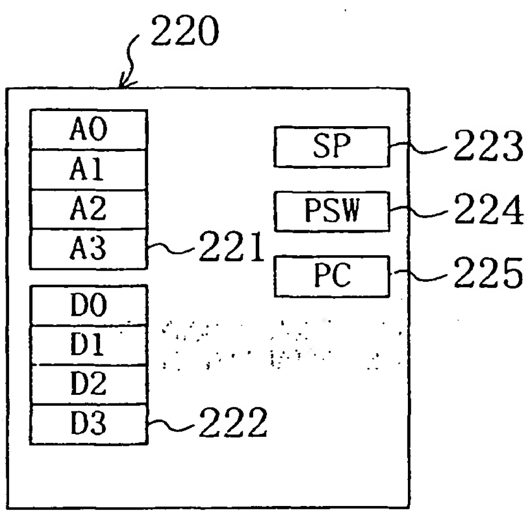


Fig. 21

NAME OF REGISTER	BIT ASSIGNMENT ON INSTRUCTION CODE	NUMBER OF PHYSICAL REGISTER	NAME OF PHYSICAL REGISTER
A0	00	00	ADDRESS REGISTER
A1	01	01	ADDRESS REGISTER
A2	02	02	ADDRESS REGISTER
A3	03	03	ADDRESS REGISTER
D0	00	00	ADDRESS REGISTER
D1	01	01	ADDRESS REGISTER
D2	02	02	ADDRESS REGISTER
D3	03	03	ADDRESS REGISTER

Fig. 22 (a)
Prior Art

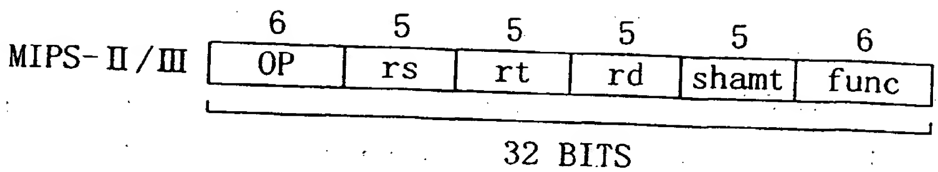


Fig. 22 (b)
Prior Art

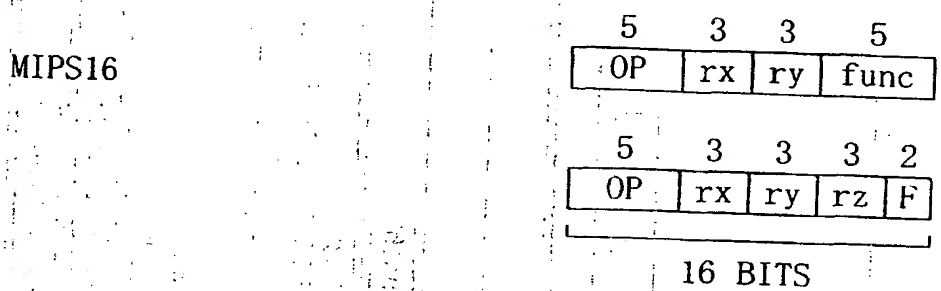


Fig. 23
Prior Art

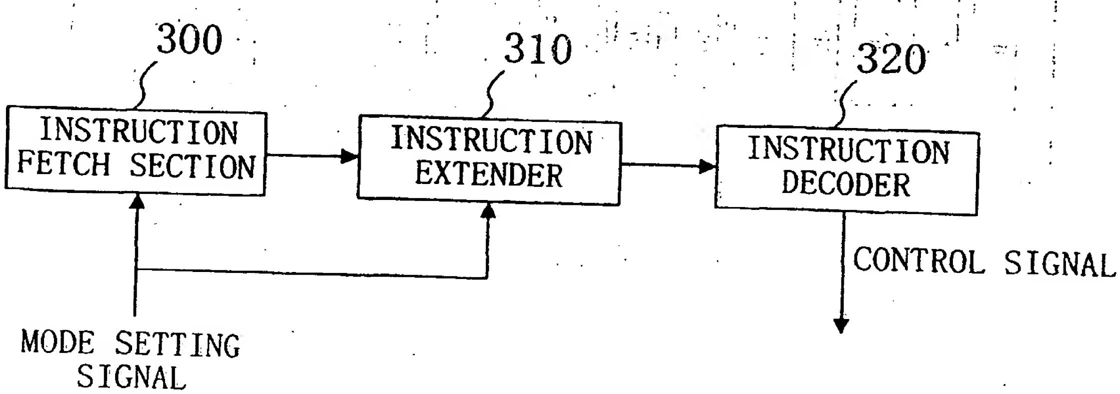


Fig. 24

FIRST INSTRUCTION FORMAT (1)	S0	OP				BIT CODES IN FIRST BYTE (HEXADECIMAL NUMBER) 00～EF
	S1	OP	imm8/d8			
	S2	OP	imm16/d16/abs16			
	S4	OP	imm32/d32/abs32			
	S6	OP	imm48			
FIRST INSTRUCTION FORMAT (2)	D0	OP	OP			F0～F4, F6
	D1	OP	OP	imm8/d8		F8
	D2	OP	OP	imm16/d16/abs16		FA
	D4	OP	OP	imm32/d32/abs32		FC
	D5	OP	OP	imm40		F5
SECOND INSTRUCTION FORMAT	T0	OP	OP	OP		F9
	T1	OP	OP	OP	imm8/d8	FB
	T3	OP	OP	OP	imm24/d24/abs24	FD
	T4	OP	OP	OP	imm32/d32/abs32	FE
THIRD INSTRUCTION FORMAT	Q0	OP	OP	OP	OP	F7

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 942 357 A3

(12) EUROPEAN PATENT APPLICATION

(88) Date of publication A3:
22.03.2000 Bulletin 2000/12

(43) Date of publication A2:
15.09.1999 Bulletin 1999/37

(21) Application number: 99104620.2

(22) Date of filing: 09.03.1999

(51) Int. Cl.[7]: **G06F 9/30**, G06F 9/318, G06F 9/38

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **11.03.1998 JP 5968098**
24.04.1998 JP 11558898

(71) Applicant:
Matsushita Electric Industrial Co., Ltd.
Kadoma-shi, Osaka 571-8501 (JP)

(72) Inventors:
- **Kishida, Takeshi**
Kashiwara-shi, Osaka 582-0028 (JP)
- **Nakajima, Masaitsu**
Osaka-shi, Osaka 536-0007 (JP)

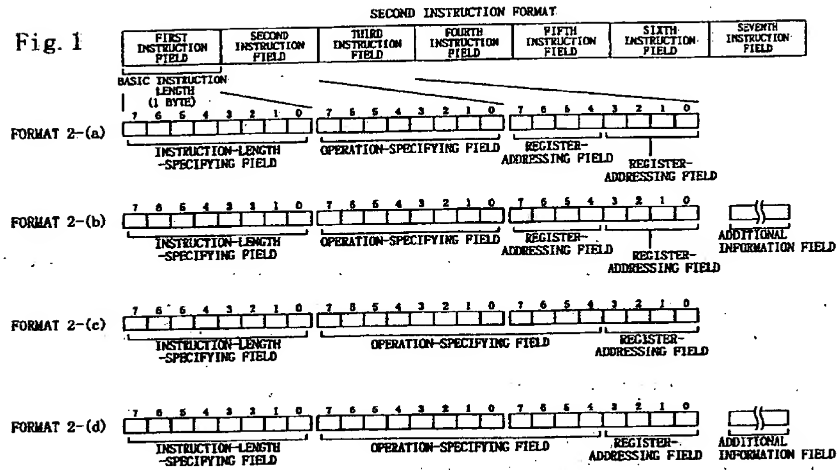
(74) Representative:
Grünecker, Kinkeldey,
Stockmair & Schwanhäusser
Anwaltssozietät
Maximilianstrasse 58
80538 München (DE)

(54) **Data processor compatible with a plurality of instruction formats**

(57) A data processor according to the present invention executes instructions described in first and second instruction formats. The first instruction format defines a register-addressing field of a predetermined size, while the second instruction format defines a register-addressing field of a size larger than that of the register-addressing field defined by the first instruction format. The data processor includes: instruction-type identifier, responsive to an instruction, for identifying the received instruction as being described in the first or second instruction format by the instruction itself; a first register file including a plurality of registers; and a second register file also including a plurality of registers, the number of the registers included in the second register file being larger than that of the registers included in the first register file. If the instruction-type identifier has identified the received instruction as being described in the first instruction format, the data processor executes the instruction using data held in the first register file. On the other hand, if the instruction-type identifier has identified the received instruction as being described in the second instruction format, the data processor executes the instruction using data held in the second register file.

EP 0 942 357 A3

Fig. 1


SECOND INSTRUCTION FORMAT
FIRST INSTRUCTION FIELD
SECOND INSTRUCTION FIELD
THIRD INSTRUCTION FIELD
FOURTH INSTRUCTION FIELD
FIFTH INSTRUCTION FIELD
SIXTH INSTRUCTION FIELD
SEVENTH INSTRUCTION FIELD
BASIC INSTRUCTION LENGTH (1 BYTE)
FORMAT 2-(a)
INSTRUCTION-LENGTH -SPECIFYING FIELD
OPERATION-SPECIFYING FIELD
REGISTER-ADDRESSING FIELD
REGISTER-ADDRESSING FIELD
FORMAT 2-(b)
INSTRUCTION-LENGTH -SPECIFYING FIELD
OPERATION-SPECIFYING FIELD
REGISTER-ADDRESSING FIELD
REGISTER-ADDRESSING FIELD
ADDITIONAL INFORMATION FIELD
FORMAT 2-(c)
INSTRUCTION-LENGTH -SPECIFYING FIELD
OPERATION-SPECIFYING FIELD
REGISTER-ADDRESSING FIELD
FORMAT 2-(d)
INSTRUCTION-LENGTH -SPECIFYING FIELD
OPERATION-SPECIFYING FIELD
REGISTER-ADDRESSING FIELD
ADDITIONAL INFORMATION FIELD

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 99 10 4620

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
X	WO 97 48041 A (LSI LOGIC CORP) 18 December 1997 (1997-12-18)	1-4, 7-14,22, 23,25	G06F9/30 G06F9/318 G06F9/38
Y	* page 10, line 17 - page 12, line 37 *	15-18,21	
Y	* page 13, line 18 - line 31 *	24	
	* page 14, line 6 - line 21 *		
	* page 20, line 32 - page 22, line 12 *		
Y	EP 0 425 410 A (IBM) 2 May 1991 (1991-05-02) * column 4, line 52 - column 6, line 40; figures 2-4 *	15-18,21	
Y	EP 0 426 393 A (FUJITSU LTD) 8 May 1991 (1991-05-08)	24	
A	* the whole document *	15-18,21	
A	"Addressing a Second Page of Registers without increasing the Register Field Length" IBM TECHNICAL DISCLOSURE BULLETIN., vol. 16, no. 3, August 1973 (1973-08), pages 771-772, XP002114028 IBM CORP. NEW YORK., US ISSN: 0018-8689 * the whole document *	1,10,22, 23	TECHNICAL FIELDS SEARCHED (Int.Cl.6) G06F
A	D. R. DITZEL ET AL.: "The Hardware Architecture of the CRISP Microprocessor" THE 14TH ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE,June 1987 (1987-06), pages 309-319, XP000212078 pittsburg, pen. US * page 310, left-hand column, paragraph 5; figure 1 *	5,6,25	

-/--

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	19 January 2000	Daskalakis, T

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 99 10 4620

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
A	EP 0 483 967 A (SUN MICROSYSTEMS INC) 6 May 1992 (1992-05-06) * the whole document *	1	
A	WO 97 22924 A (EITAN BENNY ;KOWASHI EIICHI (JP); PELEG ALEXANDER D (US); DULONG C) 26 June 1997 (1997-06-26)		
A	PATENT ABSTRACTS OF JAPAN vol. 015, no. 075 (P-1169), 21 February 1991 (1991-02-21) & JP 02 293932 A (NEC IC MICROCOMPUT SYST LTD), 5 December 1990 (1990-12-05) * abstract *	15,17	

TECHNICAL FIELDS SEARCHED (Int.Cl.6)

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	19 January 2000	Daskalakis, T

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons
& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

European Patent Office

Application Number

EP 99 10 4620

CLAIMS INCURRING FEES

The present European patent application comprised at the time of filing more than ten claims.

☐ Only part of the claims have been paid within the prescribed time limit. The present European search report has been drawn up for the first ten claims and for those claims for which claims fees have been paid, namely claim(s):

☐ No claims fees have been paid within the prescribed time limit. The present European search report has been drawn up for the first ten claims.

LACK OF UNITY OF INVENTION

The Search Division considers that the present European patent application does not comply with the requirements of unity of invention and relates to several inventions or groups of inventions, namely:

see sheet B

☒ All further search fees have been paid within the fixed time limit. The present European search report has been drawn up for all claims.

☐ As all searchable claims could be searched without effort justifying an additional fee, the Search Division did not invite payment of any additional fee.

☐ Only part of the further search fees have been paid within the fixed time limit. The present European search report has been drawn up for those parts of the European patent application which relate to the inventions in respect of which search fees have been paid, namely claims:

☐ None of the further search fees have been paid within the fixed time limit. The present European search report has been drawn up for those parts of the European patent application which relate to the invention first mentioned in the claims, namely claims:

European Patent Office

LACK OF UNITY OF INVENTION SHEET B

Application Number
EP 99 10 4620

The Search Division considers that the present European patent application does not comply with the requirements of unity of invention and relates to several inventions or groups of inventions, namely:

1. Claims: 1-14, 22,23,25,26

   Processor for executing instructions belonging to different formats

2. Claims: 15-21,24

   Exexution in parallel of compound instructions

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 99 10 4620

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report.
The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

19-01-2000

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9748041 A	18-12-1997	US 5896519 A	20-04-1999
		US 5794010 A	11-08-1998
		US 5905893 A	18-05-1999
		AU 3480897 A	07-01-1998
		GB 2329495 A	24-03-1999
EP 0425410 A	02-05-1991	US 5175863 A	29-12-1992
		JP 3138759 A	13-06-1991
EP 0426393 A	08-05-1991	JP 2835103 B	14-12-1998
		JP 3147021 A	24-06-1991
		AU 625008 B	25-06-1992
		AU 6572290 A	01-08-1991
		CA 2029088 A,C	02-05-1991
		DE 69031899 D	12-02-1998
		DE 69031899 T	16-04-1998
		KR 9307041 B	26-07-1993
		US 5442762 A	15-08-1995
EP 0483967 A	06-05-1992	CA 2054459 A,C	30-04-1992
		JP 6103062 A	15-04-1994
		KR 9601943 B	08-02-1996
		US 5771368 A	23-06-1998
WO 9722924 A	26-06-1997	US 5701508 A	23-12-1997
		AU 1430597 A	14-07-1997
		DE 19681660 T	29-10-1998
		GB 2326494 A	23-12-1998
JP 02293932 A	05-12-1990	JP 1885767 C	22-11-1994
		JP 6012522 B	16-02-1994

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82